Conflict resolution with data currency and consistency

Conflict Resolution with Data Currency and Consistency

Wenfei Fan, University of Edinburgh, UK, and RCBD and SKLSDE Lab, Beihang University, China
Floris Geerts, University of Antwerp, Belgium
Nan Tang, Qatar Computing Research Institute, Qatar
Wenyuan Yu, Facebook, Inc., USA

This paper introduces a new approach for conflict resolution: given a set of tuples pertaining to the same entity, it is to identify a single tuple in which each attribute has *the latest and consistent* value in the set. This problem is important in data integration, data cleaning and query answering. It is, however, challenging since in practice, reliable timestamps are often absent, among other things. We propose a model for conflict resolution, by specifying data currency in terms of partial currency orders and currency constraints, and by enforcing data consistency with constant conditional functional dependencies. We show that identifying data currency orders helps us repair inconsistent data, and vice versa. We investigate a number of fundamental problems associated with conflict resolution, and establish their complexity. In addition, we introduce a framework and develop algorithms for conflict resolution, by integrating data currency and consistency inferences into a single process, and by interacting with users. We experimentally verify the accuracy and efficiency of our methods using real-life and synthetic data.

Categories and Subject Descriptors: H.2 [**Database Management**]: General—*integrity*

General Terms: Theory, Algorithms, Experimentation

Additional Key Words and Phrases: Conditional functional dependency, Currency constraints, Data cleaning

1. INTRODUCTION

Conflict resolution is the process that, given a set I_t of tuples pertaining to the same entity, fuses the tuples into a single tuple and resolves conflicts among the tuples of I_t [Dong and Naumann 2009]. Traditional work resolves conflicts typically by taking, *e.g.,* the max, min, avg, any of attribute values (see [Bleiholder and Naumann 2008] for a recent survey on conflict resolution).

We study a new approach for conflict resolution, by highlighting both data currency and data consistency. Given I_t, we want to identify a single tuple in which each attribute has *consistent* and *the most current value* taken from I_t, referred to as the *true values* of the entity *relative to* I_t. The need for studying this problem is evident in data integration, where conflicts often emerge from values that refer to the same entity and come from different sources. It is also common to find multiple values of the same entity residing in a database. While these values were *once correct*, *i.e.,* they were the true values of the entity at some time, some of them may have become *stale* and thus *inconsistent*. Indeed, it is estimated that in a customer database, about 50% of the records may become obsolete within two years [Eckerson 2002]. Even in the same tuple t, some attribute values are up-to-date, while the rest may not be, due to data fusion and updates. With these comes the need for resolving conflicts for, *e.g.,* data fusion [Bleiholder and Naumann 2008; Dong and Naumann 2009], data cleaning [Arenas et al. 1999] and query answering with current values [Fan et al. 2012].

No matter how important, the problem is rather challenging. Indeed, it is already highly nontrivial to find consistent values for an entity [Arenas et al. 1999; Cong et al. 2007]. Moreover, it is hard to identify the most current entity values [Fan et al. 2012] since in the real world, reliable timestamps are often absent [Zhang et al. 2010; Goldring 1995]. Add to this the complication that when resolving conflicts one has to find the entity values that are *both* consistent *and* most current.

Fig. 1. V-J Day

	name	status	job	kids	city	AC	zip	county
E_1 r_1:	*Edith Shain*	*working*	*nurse*	*0*	*NY*	*212*	*10036*	*Manhattan*
r_2:	*Edith Shain*	*retired*	*n/a*	*3*	*SFC*	*415*	*94924*	*Dogtown*
r_3:	*Edith Shain*	*deceased*	*n/a*	null	*LA*	*213*	*90058*	*Vermont*
E_2 r_4:	*George Mendonça*	*working*	*sailor*	*0*	*Newport*	*401*	*02840*	*Rhode Island*
r_5:	*George Mendonça*	*retired*	*veteran*	*2*	*NY*	*212*	*12404*	*Accord*
r_6:	*George Mendonça*	*unemployed*	*n/a*	*2*	*Chicago*	*312*	*60653*	*Bronzeville*

Fig. 2. Instances E_1 for entity Edith and E_2 for George

Currency constraints:	φ_1: $\forall t_1, t_2$ (t_1[status] = "working" $\wedge$ t_2[status] = "retired" $\rightarrow$ $t_1 \prec_{\text{status}} t_2$)
	φ_2: $\forall t_1, t_2$ (t_1[status] = "retired" $\wedge$ t_2[status] = "deceased" $\rightarrow$ $t_1 \prec_{\text{status}} t_2$)
	φ_3: $\forall t_1, t_2$ (t_1[job] = "sailor" $\wedge$ t_2[job] = "veteran" $\rightarrow$ $t_1 \prec_{\text{job}} t_2$)
	φ_4: $\forall t_1, t_2$ (t_1[kids] $< t_2$[kids] $\rightarrow$ $t_1 \prec_{\text{kids}} t_2$)
	φ_5: $\forall t_1, t_2$ ($t_1 \prec_{\text{status}} t_2 \rightarrow t_1 \prec_{\text{job}} t_2$)
	φ_6: $\forall t_1, t_2$ ($t_1 \prec_{\text{status}} t_2 \rightarrow t_1 \prec_{\text{AC}} t_2$)
	φ_7: $\forall t_1, t_2$ ($t_1 \prec_{\text{status}} t_2 \rightarrow t_1 \prec_{\text{zip}} t_2$)
	φ_8: $\forall t_1, t_2$ ($t_1 \prec_{\text{city}} t_2 \wedge t_1 \prec_{\text{zip}} t_2 \rightarrow t_1 \prec_{\text{county}} t_2$)
Constant CFDs:	ψ_1: $\forall t$ (t[AC] $= 213 \rightarrow t$[city] $= LA$)
	ψ_2: $\forall t$ (t[AC] $= 212 \rightarrow t$[city] $= NY$)

Fig. 3. Currency constraints and constant CFDs

Example 1: The photo in Fig. 1 is known as "V-J Day in Times Square". The nurse and sailor in the photo have been identified as Edith Shain and George Mendonça, respectively, and their information is collected in sets E_1 and E_2 of tuples, respectively, shown in Fig. 2.

We want to find the true values of these entities, *i.e.,* a tuple t_1 for Edith (resp. a tuple t_2 for George) such that the tuple has the most current and consistent attribute values for her (resp. his) status, job, the number of kids, city, AC (area code), zip and county in E_1 (resp. E_2). However, the values in E_1 (E_2) have conflicts, and worse still, they do not carry timestamps. They do not tell us, for instance, whether Edith still lives in *NY*, or even whether she is still alive. Moreover, as commonly found in practice, the true values of the attributes may be scattered across different tuples and hence, we cannot find them by simply identifying a couple of tuples that are most current or consistent, *i.e.,* conflict resolution often cannot be conducted at the tuple level. □

The situation is bad, but not hopeless. We can often deduce certain currency orders from the semantics of the data. In addition, dependencies such as conditional functional dependencies (CFDs) [Fan et al. 2008] have proven useful in improving the consistency of the data. Better still, data currency and consistency interact with each other. When they are taken together, we can often infer some true values from inconsistent tuples, even in the absence of timestamps, as illustrated below.

Example 2: From the semantics of the data, we can deduce the *currency constraints* and CFDs shown in Fig. 3.

(1) Currency constraints. We know that for each person, status only changes from *working* to *retired* and from *retired* to *deceased*, but not from *deceased* to *working* or *retired*. These can be expressed as φ_1 and φ_2 given in Fig. 3, referred to as *currency constraints*. Here $t_1 \prec_{\mathsf{status}} t_2$ denotes a partial currency order defined on the attribute status, indicating that t_2 is *more current* than t_1 *in attribute* status. Similarly, we know that job can only change from *sailor* to *veteran* but not the other way around. We can express this as currency constraint φ_3, shown in Fig. 3. Moreover, the number of kids typically increases monotonically. We can express this as φ_4, assuring that t_2 is more current than t_1 in attribute kids if $t_1[\mathsf{kids}] < t_2[\mathsf{kids}]$.

In addition, we know that for each person, if tuple t_2 is more current than t_1 in attribute status, then t_2 is also more current than t_1 in job, AC and zip. Furthermore, if t_2 is more current than t_1 in attributes city and zip, it also has a more current county than t_1. These can be expressed as currency constraints φ_5–φ_8.

(2) *Constant CFDs*. In the US, if the AC is *213* (resp. *212*), then the city must be *LA* (resp. *NY*). These are expressed as conditional functional dependencies ψ_1 and ψ_2 shown in Fig. 3.

We can apply these constraints to the set E_1 of tuples given in Fig. 2, to improve the currency and consistency of the data. By *interleaving* inferences of data currency and data consistency, we can actually identify the true values of entity Edith, as follows:

(a) from the currency constraints φ_1 and φ_2, we can conclude that her latest status is *deceased*;
(b) similarly, by φ_4, we find that her true kids value is *3* (assuming null $< k$ for any number k);
(c) from (a) above and φ_5–φ_7, we know that her latest job, AC and zip are *n/a*, *213* and *90058*, respectively;
(d) after currency inferences (a) and (c), we can apply the CFD ψ_1 and find her latest city as *LA*; and
(e) after the consistency inference (d), from (c) and (d) we get her latest county as *Vermont*, by applying the currency constraint φ_8.

Now we have identified a single tuple

t_1 = (Edith Shain, deceased, n/a, 3, LA, 213, 90085, Vermont)

as the true values of Edith Shain relative to the set E_1 of tuples (the address is for her cemetery). Observe that these true values are taken from *different tuples* in E_1, *e.g.,* kids = *3* from r_2 and AC = *213* from r_3. □

This example suggests the following. (1) Data currency and consistency should be interleaved when resolving conflicts. Indeed, not only deducing currency orders helps us *improve the consistency* (*e.g.,* from steps $(a), (c)$ to (d)), but data consistency inferences also help us *identify the most current values* (*e.g.,* step (e) is doable only after (d)). (2) Both data currency and data consistency can be specified with constraints, and hence, can be processed in a uniform logical framework.

While the need for deducing the consistent and most current values has been advocated for conflict resolution [Dong and Naumann 2009; Motro and Anokhin 2006], prior work typically assumes the availability of timestamps. Previous work on data quality focuses on either data consistency (*e.g.,* [Arenas et al. 1999; Fan et al. 2008; Cong et al. 2007; Yakout et al. 2010]) or data currency (*e.g.,* [Fan et al. 2012]) taken separately.

However, no models or algorithms are yet in place to combine data consistency and currency for *conflict resolution*.

Contributions. We study conflict resolution by inferring *both* data currency *and* data consistency.

(1) We propose a model for conflict resolution (Section 2). We specify data currency in terms of (a) *partial currency orders* denoting available (yet possibly incomplete) temporal information on the data, and (b) simple *currency constraints*, to express currency relationships derived from the semantics of the data. Data consistency is specified in terms of *constant CFDs* [Fan et al. 2008] on the latest values of the data. Given such a specification S_e on a set E of tuples pertaining to the same entity e, we aim to derive the true values of e from S_e.

(2) We introduce a framework for conflict resolution (Section 3). One may find *some* true values of an entity from its specification, but *not all*, as illustrated below.

Example 3: Consider the set E_2 of tuples for entity George Mendonça (Fig. 2). Along the same lines as Example 2, we find that its true (name, kids) values are (*George Mendonça*, *2*). However, we do not have sufficient information to infer the true values of the other attributes of this entity. □

In light of this, our framework *automatically derives* as many true values as possible from a given specification S_e of an entity e, identifies attributes for which the true values of e are not derivable from S_e, and *interacts* with users to solicit additional input for those attributes, so that all the true values of all the attributes of e can be derived from S_e and users' input.

(3) We study problems fundamental to conflict resolution (Section 4). Given a specification S_e, we determine whether partial currency orders, currency constraints and CFDs in S_e have conflicts among themselves? Whether some other currency orders are implied by S_e? Whether true values of an entity can be derived from S_e? If not, what additional minimum currency information has to be provided so that the true values are derivable? We establish their complexity bounds, ranging from NP-complete and coNP-complete to Σ_2^p-complete. These results reveal the complexity *inherent to* conflict resolution.

(4) We develop several practical algorithms (Section 5). We propose methods for finding (a) whether a specification S_e has conflicts, (b) what true values can be derived from S_e, and (c) a minimum set of attributes that require users' input to find their true values. All these problems are *intractable*; in particular, the last problem is Σ_2^p-complete. Nevertheless, we provide efficient heuristic algorithms, by integrating inferences of data consistency and currency into a single process.

(5) We evaluate the accuracy and efficiency of our method using real-life and synthetic data (Section 6). We find that *unifying* currency and consistency *substantially improves* the accuracy of traditional methods, by 201% (F-measure), even with only a small number of constraints. It is also more effective than taking consistency and currency *separately*. Furthermore, our algorithms are efficient, and scale well with the number of tuples pertaining to an entity and with the number of constraints; for example, it takes an average of 7 seconds to resolve conflicts in sets of 8k-10k tuples representing an entity, with 1983 constraints.

We contend that this work provides fundamental results for conflict resolution, and propose a practical solution via inferences of data currency and data consistency in the *absence* of timestamps.

Related work. This work extends [Fan et al. 2013] by including (1) a comprehensive analysis of the fundamental problems in connection with conflict resolution in terms of currency constraints and constant CFDs (Section 4); and (2) a detailed discussion of algorithms and procedures needed for finding the most current values (Section 5). None of the proofs of (1) was presented in [Fan et al. 2013]. These proofs are interesting in their own right. Most procedures of (2) were not given in [Fan et al. 2013].

Conflict resolution has been studied for decades, started from [Dayal 1983]. It aims to combine data from different sources into a single representation (see [Bleiholder and Naumann 2008; Dong and Naumann 2009] for surveys). In that context, inconsistencies are typically resolved by selecting the max, min, avg, any value [Bleiholder and Naumann 2008]. While the need for data currency was also observed there (*e.g.,* [Dong and Naumann 2009; Motro and Anokhin 2006]), previous work identifies current values only by using *timestamps*. This work differs from the traditional work in the following. (1) We revise the conflict resolution problem to identify values of entities that are both *consistent* and most *current*. (2) We *do not* assume the availability of timestamps, which are often missing in practice [Zhang et al. 2010; Goldring 1995]. (3) We resolve conflicts by using currency constraints and constant CFDs [Arenas et al. 1999; Fan et al. 2008; Cong et al. 2007], instead of picking max, min, avg or any value. (4) We employ *automated reasoning* to identify true values by unifying the inferences of data currency and consistency.

There has been work on truth discovery from data sources [Dong et al. 2009; Galland et al. 2010; Yin et al. 2008]. Their approaches include (1) vote counting and probabilistic computation based on the trustworthiness of data sources [Galland et al. 2010; Yin et al. 2008]; (2) source dependencies to find copy relationships and reliable sources [Dong et al. 2009]; and (3) employing lineage information and probabilities [Widom 2005]. In contrast, we assume no information about the accuracy of data sources, but derive true values based on data currency and consistency. In addition, we adopt a logical approach via automated reasoning about constraints, as opposed to probabilistic computation. This work is complementary to the previous work and can be combined with the prior approaches.

This work extends [Fan et al. 2012; Fan et al. 2008]. A data currency model was presented in [Fan et al. 2012] with partial currency orders and denial constraints [Arenas et al. 1999]. CFDs were studied for specifying data consistency [Fan et al. 2008]. This work differs from [Fan et al. 2012; Fan et al. 2008] in the following. (1) We propose a conflict resolution model that combines data currency and consistency. In contrast, [Fan et al. 2012] only studies data currency, while [Fan et al. 2008] only considers data consistency. (2) We *interleave* inferences of data currency and consistency, which is far more intriguing than handling currency and consistency separately, and requires new techniques to capture the interaction between the two. (3) We use currency constraints, which are simpler than denial constraints, to strike a balance between the complexity of inferring true values and the expressivity needed for specifying currency (Section 4). (4) *No practical algorithms* were given in [Fan et al. 2012] for deriving current values.

Previous work on data consistency [Arenas et al. 1999; Fan et al. 2008; Cong et al. 2007; Yakout et al. 2010; Greco et al. 2003; Dallachiesa et al. 2013] has been focusing on consistent query answering and data repairing [Bertossi 2011], topics different from conflict resolution (see [Fan and Geerts 2012] for a recent survey on data consistency). The study of preferred repairs [Greco et al. 2003] also advocates partial orders. It dif-

fers from the currency orders we study here in that they use PTIME functions to rank different repairs over the entire database, whereas we derive the currency orders by automated *reasoning about both* available partial temporal information and currency constraints. Preferred repairs are implemented by [Cong et al. 2007] via a cost metric, and by [Yakout et al. 2010] based on a decision theory, which can be incorporated into our framework.

There has been a large body of work on temporal databases (see [Chomicki and Toman 2005] for a survey). In contrast to that line of work, we do not assume the availability of timestamps. It has also recently been shown that temporal information helps record linkage identify records that refer to the same entity [Li et al. 2011]. Here we show that data currency helps conflict resolution as well, a different process that takes place *after* record linkage has identified tuples pertaining to the same entity. While [Li et al. 2011] is based on timestamps, we do not assume it here.

Organization. The rest of the paper is organized as follows. We propose a model for specifying conflicts in Section 2, based on data currency and consistency, and introduce a framework for resolving conflicts in Section 3. Problems fundamental to conflict resolution are studied in Section 4, and practical algorithms underlying the conflict resolution framework are developed in Section 5. An experimental study is reported in Section 6, followed by directions for future work in Section 7.

2. A CONFLICT RESOLUTION MODEL

In this section we show how to capture conflicts in terms of data currency and data consistency. We start with data currency (Section 2.1) and consistency (Section 2.2) specifications. We then present our model for characterizing conflicts commonly found in the real world by means of the specifications (Section 2.3).

2.1. Data Currency

We specify the currency of data by means of (a) partial currency orders, and (b) currency constraints.

Data with partial currency orders. Consider a relation $R = (A_1, \ldots, A_n)$, where each attribute A_i has a domain $\mathsf{dom}(A_i)$. In this work we focus on *entity instances* I_e of R, which are sets of tuples of R *all* pertaining to the *same* real-world entity e, and are typically much smaller than a database instance. Such entity instances can be identified by *e.g.,* record linkage techniques (see [Elmagarmid et al. 2007] for a survey). For an attribute $A_i \in R$ and an entity instance I_e of R, we denote by $\mathsf{adom}(I_e.A_i)$ the set of A_i-attribute values that occur in I_e, referred to as *the active domain of* A_i *in* I_e.

We have seen two entity instances given in Fig. 2: $E_1 = \{r_1, r_2, r_3\}$ for entity "Edith", and $E_2 = \{r_4, r_5, r_6\}$ for "George". Here $\mathsf{adom}(E_1.\mathsf{city})$ = {*NY, SFC, LA*}; similarly for other attributes.

A *temporal instance* I_t of I_e is given as $(I_e, \preccurlyeq_{A_1}, \ldots, \preccurlyeq_{A_n})$, where each $\preccurlyeq_{A_i}$ is a partial order on I_e, referred to as the *currency order for attribute* A_i for the entity e represented by I_e. For all $t_1, t_2 \in I_e$, $t_1 \preccurlyeq_{A_i} t_2$ if and only if (iff) either t_1 and t_2 share the same A_i-attribute value (*i.e.,* $t_1[A_i] = t_2[A_i]$), or $t_2[A_i]$ is more current than $t_1[A_i]$ (denoted by $t_1 \prec_{A_i} t_2$).

Intuitively, currency orders represent *available* temporal information about the data. Observe that $\preccurlyeq_{A_i}$ is a *partial order*, possibly empty. For example, for E_1 above, we only know that $r_3 \preccurlyeq_{\mathsf{kids}} r_1$ and $r_3 \preccurlyeq_{\mathsf{kids}} r_2$ since $r_3[\mathsf{kids}]$ is null, which are in the currency order $\preccurlyeq_{\mathsf{kids}}$, while the currency orders for other attributes are empty, excluding the case when tuples carry the same attribute value. Similarly for E_2. In particular,

$t_1 \preccurlyeq_{A_i} t_2$ if $t_1[A_i]$ is null, *i.e.,* an attribute with value missing is ranked the lowest in the currency order.

Current instances. Currency orders are often incomplete. Hence we consider possible completions of currency orders. More specifically, a *completion* I_t^c of I_t is a temporal instance $I_t^c = (I_e, \preccurlyeq^c_{A_1}, \ldots, \preccurlyeq^c_{A_n})$, such that for each $i \in [1, n]$,

(1) $\preccurlyeq_{A_i} \subseteq \preccurlyeq^c_{A_i}$, and
(2) for all tuples $t_1, t_2 \in I_e$, either $t_1 \preccurlyeq^c_{A_i} t_2$ or $t_2 \preccurlyeq^c_{A_i} t_1$.

That is, $\preccurlyeq^c_{A_i}$ induces a *total order* on the A_i attribute values in the tuples of I_e. Intuitively, I_t^c totally sorts the attribute values in I_e such that the most current value of each attribute is the last in the order.

We define *the most current A_i-attribute value of* I_t^c to be $t[A_i]$ that comes last in the total order $\preccurlyeq^c_{A_i}$. The *current tuple* of I_t^c, denoted by $\mathsf{LST}(I_t^c)$ (*i.e.,* last), is defined to be the tuple t_l such that for each attribute A_i, $t_l[A_i]$ is the most current A_i-value of I_t^c, *i.e.,* t_l contains the most current values from I_t^c.

Currency constraints. In addition to partial currency orders, we can derive additional currency information from the semantics of the data modeled as *currency constraints*. A currency constraint φ is of the form

$$\forall t_1, t_2\ (\omega \rightarrow t_1 \prec_{A_r} t_2),$$

where ω is a conjunction of predicates of the form:

(1) $t_1 \prec_{A_l} t_2$, *i.e.,* t_2 is more current than t_1 in attribute A_l;
(2) $t_1[A_l]$ op $t_2[A_l]$, where op is $=, \neq, >, <, \leq$ or $\geq$; and
(3) $t_i[A_l]$ op c for $i \in \{1, 2\}$, where c is a constant.

In contrast to denial constraints adopted in the model of [Fan et al. 2012] that was define on an unbounded number of tuples, currency constraints are defined on two tuples, like functional dependencies. Such constraints suffice to specify currency information commonly found in practice (see, *e.g.,* Example 2).

Currency constraints are interpreted over completions I_t^c of I_t. We say that I_t^c *satisfies* φ, denoted by $I_t^c \models \varphi$, if for any two tuples t_1, t_2 in I_e, if these tuples and related order information in I_t^c satisfy the predicates in ω, following the standard semantics of first-order logic, then $t_1 \prec^c_{A_r} t_2$. We say that I_t^c satisfies a set Σ of currency constraints, denoted by $I_t^c \models \Sigma$, if $I_t^c \models \varphi$ for all $\varphi \in \Sigma$.

Example 4: Recall entity instances E_1 and E_2 from Fig. 2. Currency constraints on E_1 and E_2 include φ_1–φ_8 as specified in Fig. 3 and interpreted in Example 2.

It is readily verified that for any completion E_1^c of E_1, if it satisfies these constraints, it yields $\mathsf{LST}(E_1^c)$ of the form (*Edith*, *deceased*, *n/a*, *3*, x_{city}, *213*, *90058*, x_{county}) for Edith, in which the most current values for attributes name, status, job, kids, AC and zip are deduced from the constraints and remain unchanged, while x_{city} and x_{county} are values determined by the total currency order given in E_1^c. As remarked earlier, the values of the current tuple come from *different tuples* in E_1, *e.g.,* r_2 and r_3.

Similarly, for any completion of E_2, its current tuple has the form (*George*, x_{status}, x_{job}, *2*, $x_{\mathsf{city}}, x_{\mathsf{AC}}, x_{\mathsf{zip}}, x_{\mathsf{county}}$), if they satisfy all the constraints. From these we can see that currency constraints help us find the most current values of some attributes, but not necessarily *all attributes*. □

2.2. Data Consistency

To specify the consistency of data in our conflict model, we use a simple class of conditional functional dependencies (CFDs) [Fan et al. 2008], known as constant CFDs. A *constant* CFD [Fan et al. 2008] ψ is of the form

$$\forall t\ (\nu \rightarrow t[A_r] = c_r),$$

where c_r is a constant from $\mathsf{dom}(A_r)$ and ν is a conjunction of predicates of the form $t[A_l] = c_l$, for $c_l \in \mathsf{dom}(A_l)$. For example, ψ_1 and ψ_2 in Fig. 3 are constant CFDs, as interpreted in Example 2.

Such CFDs are defined on *the current tuple* of a completion. Consider a completion I_t^c of I_t, for which the current tuple is $t_l = \mathsf{LST}(I_t^c)$. We say that the completion I_t^c *satisfies* a ψ, denoted by $I_t^c \vDash \psi$, if whenever t_l satisfies the predicate ν, following the standard semantics of first-order logic, then $t_l[A_r] = c_r$.

Intuitively, this assures that if t_l agrees with the constants specified in ν and if these are most current, then $t_l[A_r]$ should take value c_r, and $t_l[A_r]$ is the most current value in attribute A_r.

We say that I_t^c *satisfies* a set Γ of constant CFDs, denoted by $I_t^c \vDash \Gamma$, iff $I_t^c \vDash \psi$ for each $\psi \in \Gamma$.

Observe that a constant CFD is defined on a *single tuple* $\mathsf{LST}(I_t^c)$. In light of this, we do not need general CFDs of [Fan et al. 2008], which are typically defined on *two tuples*.

Example 5: Recall the current tuples for E_1 described in Example 4. Then all completions of E_1 that satisfy ψ_1 in Fig. 3 have the form (*Edith*, *deceased*, *n/a*, *3*, *LA*, *213*, *90058*, *Vermont*), in which x_{city} is instantiated as *LA* to be ψ_1, and as a result, x_{county} then becomes *Vermont* by the currency constraint φ_8. □

2.3. Conflict Resolution

We are ready to specify entities.

Specifications. A *specification* $S_e = (I_t, \Sigma, \Gamma)$ *of an entity* consists of
(1) a temporal instance $I_t = (I_e, \preccurlyeq_{A_1}, \ldots, \preccurlyeq_{A_n})$;
(2) a set Σ of currency constraints; and
(3) a set Γ of constant CFDs.
A completion $I_t^c = (I_e, \preccurlyeq^c_{A_1}, \ldots, \preccurlyeq^c_{A_n})$ of I_t is called a *valid completion* of specification S_e if I_t^c satisfies both Σ and Γ.

We say that S_e is *valid* if there exists a valid completion I_t^c of S_e. For instance, the specification of E_1 (or E_2) with the constraints given in Fig. 3 is valid.

True values. There may be many valid completions I_t^c, each leading to a possibly different current tuple $\mathsf{LST}(I_t^c)$. When two current tuples differ in some attribute, there is *a conflict*. We aim to resolve such conflicts. If all such current tuples agree on *all* attributes, then the specification is conflict-free, and a *unique* current tuple exists for the entity e specified by S_e. In this case, we say that this tuple is the true value of e.

The *true value* of S_e, denoted by $\mathsf{T}(S_e)$, is the *single* tuple t_c such that for *all valid* completions I^c of S_e, $t_c = \mathsf{LST}(S_e)$, if it exists. For each attribute A_i of R, we call $t_c[A_i]$ the *true value* of A_i in specification S_e.

The conflict resolution problem. Consider a specification $S_e = (I_t, \Sigma, \Gamma)$ of an entity e, where $I_t = (I_e, \preccurlyeq_{A_1}, \ldots, \preccurlyeq_{A_n})$. Given S_e, conflict resolution is to find the minimum amount of additional currency information such that the true value exists.

The additional currency information is specified in terms of a *partial temporal order* $O_t = (I, \preccurlyeq'_{A_1}, \ldots, \preccurlyeq'_{A_n})$. We use $S_e \oplus O_t$ to denote the extension $S'_e = (I'_t, \Sigma, \Gamma)$ of S_e by enriching I_t with O_t, where $I'_t = (I_e \cup I, \preccurlyeq_{A_1} \cup \preccurlyeq'_{A_1}, \ldots, \preccurlyeq_{A_n} \cup \preccurlyeq'_{A_n})$. In the sequel we only consider partial temporal orders O_t such that $\preccurlyeq_{A_i} \cup \preccurlyeq'_{A_i}$ is a partial order for all $i \in [1, n]$.

I_e	an entity instance of relation schema R
I_t	a temporal instance of I_e (partial currency orders)
I_t^c	a completion of partial currency orders in I_t
S_e	(I_t, currency constraints Σ, constant CFDs Γ)
$\mathsf{LST}(I_t^c)$	the current tuple of a completion I_t^c
O_t	a partial temporal order
$\mathsf{T}(S_e)$	the true values of the entity specified by S_e

Fig. 4. A summary of notations

We use $|O_t|$ to denote $\Sigma_{i\in[1,n]}|\preccurlyeq'_{A_i}|$, *i.e.,* the sum of the sizes of all the partial orders in O_t (note that each $\preccurlyeq'_{A_i}$ is a binary relation, and $|\preccurlyeq'_{A_i}|$ is its cardinality).

Given a valid specification $S_e = (I_t, \Sigma, \Gamma)$ of an entity e, the *conflict resolution problem* is to find a partial temporal order O_t such that
- the true value $\mathsf{T}(S_e \oplus O_t)$ of e exists and
- $|O_t|$ is minimum.

Example 6: Recall from Example 4 the current tuples for George. Except for name and kids, we do not have a unique current value for the other attributes. Nonetheless, if a partial temporal order O_t with, *e.g.,* $r_6 \prec_{\mathsf{status}} r_5$ is provided by the users (*i.e.,* status changes from *unemployed* to *retired*), then the true value of George in E_2 can be derived as (*George*, *retired*, *veteran*, *2*, *NY*, *212*, *12404*, *Accord*) from the currency constraints and the constant CFDs of Fig. 3. □

We summarize the notations in Fig. 4.

3. A CONFLICT RESOLUTION FRAMEWORK

We propose a framework for conflict resolution. As depicted in Fig. 5, given a specification $S_e = (I_t, \Sigma, \Gamma)$ of an entity e, the framework is to find the true value $\mathsf{T}(S_e)$ of e by reasoning about data currency and consistency, and by interacting with the users to solicit additional data currency information.

The framework provides the users with suggestions. A *suggestion* is a minimum set $\mathcal{A}$ of attributes of e such that if the true values of these attributes are provided by the users, $\mathsf{T}(S_e)$ can be automatically deduced from the users' input, Σ, Γ and I_t. The true values for $\mathcal{A}$ are represented as a temporal order O_t.

More specifically, the framework deduces $\mathsf{T}(S_e)$ as follows.

(1) *Validity checking.* It first inspects whether $S_e \oplus O_t$ is valid, via automated reasoning, where O_t is a partial temporal order provided by the users, *initially empty* (see step (4) below for details about O_t). If valid, it follows the "Yes" branch. Otherwise the users need to revise O_t by following the "No" branch.

(2) *True value deducing.* After $S_e \oplus O_t$ is validated, it derives as many true values for the attributes of e as possible, via automated reasoning.

(3) *Finding the true value*. If $\mathsf{T}(S_e \oplus O_t)$ exists, it computes and returns it following the "Yes" branch. Otherwise, it follows the "No" branch and goes to step (4).

(4) *Generating suggestions.* It computes a suggestion $\mathcal{A}$ along with its candidate values taken from the active domain of S_e, such that if the users pick and validate the true values for $\mathcal{A}$, then $\mathsf{T}(S_e \oplus O_t)$ is warranted to be found. The users are expected to provide V, the true values of *some attributes* in $\mathcal{A}$, represented as a partial temporal

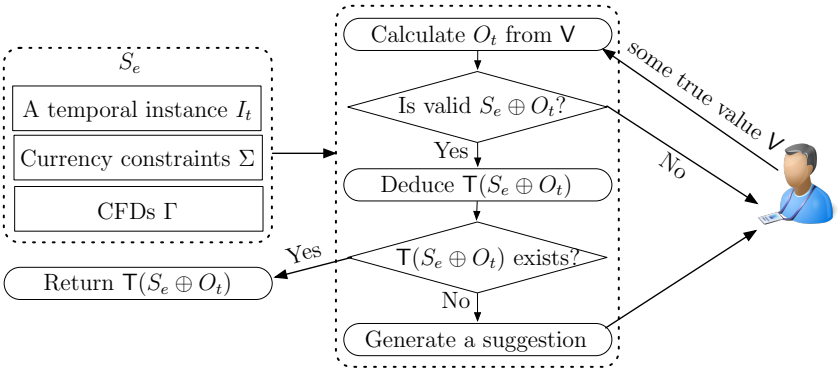


Fig. 5. Framework overview

order O_t. Given O_t, $S_e \oplus O_t$ is constructed and the process goes back to step (1).

The process proceeds until $\mathsf{T}(S_e \oplus O_t)$ is found, or when the users opt to settle with true values for a subset of attributes of e. That is, if users do not have sufficient knowledge about the entity, they may let the system derive true values for as many attributes as possible, and revert to the traditional methods to pick the max, min, avg, any values for the rest of the attributes.

By leveraging user input, the proposed conflict resolution framework can thus be applied to relations consisting of arbitrarily many attributes and specifications that do not necessarily allow to infer the current values of all attributes. Indeed, suppose that an attribute A is not covered by any constraint or any partial order, then we will put it in the suggestion $\mathcal{A}$. In other words, the true value of A has to be provided by the users. We experimentally verify the amount of user interaction needed to resolve conflicts in Section 6.

Remarks. (1) To specify users' input, let I_t in S_e be $(I_e, \preccurlyeq_{A_1}, \ldots, \preccurlyeq_{A_n})$ and $\mathcal{A} \cup \mathcal{A}' \cup \mathcal{B}$ = $\{A_1, \ldots, A_n\}$, where (i) $\mathcal{A}$ is the set of attributes identified in step (4) for which the true values are unknown; (ii) for $\mathcal{B}$, their true values $\mathsf{V}_\mathcal{B}$ have been deduced (step (2)); and (iii) $\mathcal{A}'$ is the set of attributes whose true values can be deduced from $\mathsf{V}_\mathcal{B}$ and the suggestion for $\mathcal{A}$. Given a suggestion, the user is expected to provide a set V of true values for (a subset of) $\mathcal{A}$ that they are confident of. Here V consists of either the candidate values taken from the suggestion, or some *new* values not in the active domains of S_e that users opt to choose. The users *do not* have to enter values for *all* attributes in $\mathcal{A}$.

From the input V, a partial temporal order O_t is automatically derived, by treating V as the most current values of those attributes involved. Indeed, O_t has the form $(I_e \cup \{t_o\}, \preccurlyeq'_{A_1}, \ldots, \preccurlyeq'_{A_n})$, where t_o is a new tuple such that for all attributes A, $t_o[A] = \mathsf{V}(A)$ if V has a value $\mathsf{V}(A)$ for A, and $t_o[A] = \mathsf{null}$ otherwise, while $t_o[\mathcal{B}] = \mathsf{V}_\mathcal{B}$ remains unchanged. Moreover, $\preccurlyeq'_A$ extends $\preccurlyeq_A$ by including $t[A] \preccurlyeq'_A t_o[A]$ if $t_o[A] \neq \mathsf{null}$, for all tuples $t \in I_e$. From this, $S_e \oplus O_t$ can be readily defined.

(2) The framework requires currency constraints and constant CFDs as input. There have been efficient methods for discovering constant CFDs, *e.g.,* [Chiang and Miller 2008; Fan et al. 2011]. Along the same lines as CFD discovery, automated methods can be developed for discovering currency constraints from (possibly dirty) data.

Below we outline an alternative approach to discovering currency constraints, by capitalizing on mining methods for association rules [Calders et al. 2006]. The idea is to transform the data in an entity instance I_e into a so-called transaction database in which the attribute values are Boolean. A wide variety of rule discovery methods are already in place on such kind of databases (see [Goethals 2003] for a survey).

In a nutshell, the database transformation consists of two steps. (1) For each attribute A_i of R we create a new set of attributes consisting of $A_{(\_,\_)}$, $A_{(\_,a)}$ and $A_{(a,\_)}$ for all frequent values $a \in dom(A_i)$ that occur in I_e. By setting an appropriate frequency threshold, the number of newly created attributes can be limited. (2) For each pair of tuples $t_1, t_2 \in I$ we then define a $0/1$-tuple t_{12} over the new set of attributes by letting $t_{12}[A_{(\_,\_)}] = 1$ if $t_1 \prec_A t_2$; $t_{12}[A_{(\_,a)}] = 1$ if $t_1 \prec_A t_2$ and $t_2[A] = a$; and $t_{12}[A_{(a,\_)}] = 1$ if $t_1[A] = a$ and $t_1 \prec_A t_2$. Otherwise, these attributes are set to 0. In other words, t_{12} encodes which partial order relationships hold between tuple t_1 and t_2.

It can now be easily seen that an association rule over the transaction database corresponds to a currency constraint on I_e and vice versa. We leave the experimental validation and design of more efficient currency discovery algorithms to future work.

(3) To simplify the discussion we do not allow users to change constraints in S_e. We defer further discussion about this to Section 7.

(4) We assume that the values in entity instances were once correct. When a temporal instance contains errors, one may inspect different samples and only take those currency orders that are consistent among the samples or have sufficient support (*e.g.,* frequency).

4. FUNDAMENTAL PROBLEMS

In this section, we identify fundamental problems associated with conflict resolution based on both data currency and consistency, and establish their complexity. These results are not only of theoretical interest, but also tell us where the complexity arises, and hence guide us to develop effective (heuristic) algorithms.

Satisfiability. The first one is the *satisfiability problem* for entity specifications. It is to decide, given a specification $S_e = (I_t, \Sigma, \Gamma)$ of an entity, whether S_e is *valid*, *i.e.,* whether there exists a valid completion of S_e.

Intuitively, it is to check whether S_e makes sense, *i.e.,* whether the currency constraints, constant CFDs and partial orders in S_e, when put together, have conflicts themselves. The analysis is needed in step (1) of the framework of Fig. 5. In practice, this analysis tells us whether we have to revise constraints in S_e, or ask users to validate its partial orders.

The problem is obviously important, but is NP-complete. One might think that the absence of currency constraints or CFDs would simplify the analysis. Unfortunately, its intractability is rather *robust*.

THEOREM 4.1. *The satisfiability problem for entity specifications is NP-complete. It remains NP-hard for valid specifications $S_e = (I_t, \Sigma, \Gamma)$ of an entity when (1) both Σ and Γ are fixed; (2) $\Gamma = \emptyset$,* i.e., *when only currency constraints are present; or (3) $\Sigma = \emptyset$,* i.e., *when only constant CFDs are present.*

Proof: For the upper bound it suffices to observe that the following NP algorithm correctly decides whether a given specification has a valid completion. Given a specification $S_e = (I_t, \Sigma, \Gamma)$, the algorithm simply guesses a completion I_t^c of I_t and then checks whether (i) $I_t^c \models \Sigma$; and (ii) $I_t^c \models \Gamma$. If the guessed completion passes these checks, then

the algorithm returns "yes". Otherwise, the guessed completion is rejected. Note that a "guess" simply completes the partial orders on the values of I_t, and there are finitely many guesses in total. The algorithm is in NP since checking can be done in PTIME.

The NP-lower bound is established by reduction from the 3-satisfiability problem. An instance of the 3-satisfiability problem is formula $\varphi = C_1 \wedge \cdots \wedge C_r$ with $C_j = \ell_1^j \vee \ell_2^j \vee \ell_3^j$, where for $k \in \{1, 2, 3\}$ and $j \in [1, r]$, ℓ_k^j is either a variable or a negation of a variable from a set $X = \{x_1, \ldots, x_n\}$ of variables. It is to determine whether φ is satisfiable, *i.e.,* whether there exists a truth assignment of variables in X that satisfies φ. This problem is known to be NP-complete (cf. [Papadimitriou 1994]).

Given φ, we define a specification $S_e = (I_t, \Sigma, \Gamma)$ such that there exists a valid completion of S_e if and only if φ is satisfiable. The specification S_e consists of a temporal instance I_t of schema $R(D, C, P, U, V, W)$ and a fixed set of currency constraints Σ. No constant CFDs are defined in S_e. Intuitively, D is to distinguish between tuples that encode truth assignments and tuples that correspond to clauses in φ; C is to identify variables (by x_i) and clauses (by $j \in [1, r]$); P is used to enforce the validity of clauses and finally, U, V and W represent the positions (1, 2 and 3, resp) of variables in each clause.

We first explain how the temporal instance I_t of R together with the currency constraints in Σ is to encode truth assignments for X and clauses in φ. More specifically, for each variable $x_i \in X$, we use two constants a_i and b_i such that $a_i \preccurlyeq_A b_i$ encodes that x_i is set to true, whereas $b_i \preccurlyeq_A a_i$ encodes that $\bar{x}_i$ is set to true (or, equivalently that x_i is set to false). Here A ranges over attributes U, V and W. More specifically, for each variable $x_i \in X$ we include two tuples in I_t:

$$(0, x_i, 0, a_i, a_i, a_i) \quad \text{and} \quad (0, x_i, 0, b_i, b_i, b_i).$$

These encode truth assignments of X. To ensure that the choice of truth value for variables is consistent, we include the following currency constraints in Σ:

$$\forall t_1, t_2\ (t_1[D] = 0 \wedge t_2[D] = 0 \wedge t_1[C] = t_2[C] \wedge t_1[A] \prec t_2[A] \rightarrow t_1[B] \prec t_2[B]),$$

where A and B range over distinct pairs taken from $\{U, V, W\}$. These currency constraints enforce that variables x_i are set to true (resp. false) independent of the position at which they appear in clauses (*i.e.,* in attribute U, V or W).

We next consider the clauses in φ. Let $C_j = \ell_1^j \vee \ell_2^j \vee \ell_3^j$. observe that this can be equivalently written as $\bar{\ell}_1^j \wedge \bar{\ell}_2^j \rightarrow \ell_3^j$. For instance, consider a clause $C = x_1 \vee \bar{x}_2 \vee \bar{x}_3$. This is equivalent to $\bar{x}_1 \wedge x_2 \rightarrow \bar{x}_3$. Given this, we include two tuples in I_t for each clause C_j:

$$(1, j, 1, v_1, v_2, v_3) \quad \text{and} \quad (1, j, 2, v_1', v_2', v_3'),$$

where $v_i = a_k$ and $v_i' = b_k$ if $\ell_i^j = \bar{x}_k$, and $v_i = b_k$ and $v_i' = a_k$ if $\ell_i^j = x_k$, for $i = 1, 2$, and conversely for $i = 3$. The example clause C is thus encoded by $(1, \_, 1, b_1, a_2, b_3)$ and $(1, \_, 2, a_1, b_2, a_3)$. The connection between truth assignments selected by completions and the validity of clauses is established by means of the following currency constraint:

$$\forall t_1, t_2\ (t_1[D] = 1 \wedge t_2[D] = 1 \wedge t_1[C] = t_2[C] \wedge t_1[P] = 1 \wedge t_2[P] = 2 \\ \wedge t_1[U] \prec t_2[U] \wedge t_1[V] \prec t_2[V] \rightarrow t_1[W] \prec t_2[W]).$$

This constraint tells us that whenever the truth assignment (represented by a completion) makes $\bar{\ell}_1^j \wedge \bar{\ell}_2^j$ true, then it must also make ℓ_3^j true.

We next show the correctness of the reduction. Suppose that φ is true and let μ_X be a satisfying truth assignment. We define a valid completion of S_e as follows. For attributes D, C and P we order the tuples in I_t arbitrarily. For attributes U (and consequently also for V and W by the currency constraints) we set $a_i \preccurlyeq_U^c b_i$ if $\mu_X(x_i)$ is

true, and $b_i \preccurlyeq^c_U a_i$ otherwise. We need to verify that the second currency constraint is satisfied. This follows immediately from the fact that each clause is satisfied by μ_X. Conversely, suppose that we have a valid completion of S_e. From this, we define μ_X by simply setting $\mu_X(x_i) = 1$ if $a_i \preccurlyeq^c_U b_i$ and $\mu_X(x_i) = 0$ otherwise. Similarly as above, it is readily verified that μ_X satisfies all the clauses. Indeed, this follows from the second currency constraint given above.

It remains to show that the satisfiability problem is NP-complete when (1) Σ and Γ are fixed; (2) $\Gamma = \emptyset$; or (3) $\Sigma = \emptyset$. Since we have shown that the satisfiability problem is in NP, for general Σ and Γ, it suffices to show the lower bounds. Furthermore, observe the proof above uses (i) a fixed set of currency constraints, *i.e.,* the currency constraints Σ are independent of the input instance φ, and (ii) it does not use any constant CFDs. In other words, (1) and (2) follow directly from the lower bound proof given above.

It remains to show (3), *i.e.,* the satisfiability problem is NP-hard even when only constant CFDs are present. We establish this lower bound by reduction from the complement of the tautology problem, which is known to be coNP-complete (cf. [Papadimitriou 1994]). An instance of the tautology problem is a formula $\varphi = C_1 \vee \cdots \vee C_r$, where $C_j = \ell^j_1 \wedge \ell^j_2 \wedge \ell^j_3$ and each ℓ^j_k is either a variable or a complement of a variable from $X = \{x_1, \ldots, x_n\}$. It is to determine whether φ is true for all truth assignments of X. We define a specification $S_e = (I_t, \Sigma = \emptyset, \Gamma)$ such that S_e has a valid completion if and only if φ is not a tautology.

The temporal instance I_t of S_e is an instance of schema $R'(X_1, \ldots, X_n, C)$; it consists of two tuples $(0, 0, \ldots, 0)$ and $(1, 1, \ldots, 1)$. We impose no currency order or currency constraints on I_t. Note that each completion I^c_t yields a current tuple $\mathsf{LST}(I^c_t)$ that encodes a truth assignment μ_X of X in its first n attributes.

The set Γ of constant CFDs is given as follows. For each clause C_j, we define ψ_j:

$$\forall t\ (t[L_1] = c_1 \wedge t[L_2] = c_2 \wedge t[L_3] = c_3 \rightarrow t[C] = 1),$$

where $L_i = X_k$ if ℓ^j_i or $\bar{\ell}^j_i$ is x_k and $c_i = 1$ if $\ell^j_i = x_k$ and $c_i = 0$ if $\ell^j_i = \bar{x}_k$, for $i = 1, 2, 3$. Clearly, a completion $I^c_t \models \psi_j$ if the truth assignment μ_X encoded by the current tuple $\mathsf{LST}(I^c_t)$ makes C_j true. We further add the CFD $\psi_C = \forall t\ (t[C] = 1 \rightarrow t[C] = 0)$ to Γ, which intuitively prevents any clause to be satisfied. Indeed, a completion I^c_t such that $I^c_t \models \psi_C$ must set the C-attribute of its current tuple to 0. Contrast this with the requirement on the C-attribute of current tuples imposed by the ψ_j's.

We next show the correctness of the reduction. If φ is a tautology then every truth assignment μ_X makes at least one clause C_j true. That is, any valid I^c_t must set the C-attribute of its current tuple to 1 (by ψ_j) and at the same time it must set the C-attribute of its current tuple to 0 (by ψ_C). Hence, no valid completion can exists. Conversely, if there exists a valid completion I^c_t of S_e such that $I^c_t \models \Gamma$, then its current tuple must have its C-attribute set to 0. In other words, none of the left-hand sides of the ψ_j's can be true, and hence μ_X must make all clauses false. In other words, μ_X is a counterexample to the validity of φ and hence φ is not a tautology. □

Implication. The second problem aims to deduce partial temporal orders that are logical consequences of the given currency order and currency constraints. Consider a valid specification $S_e = (I_t, \Sigma, \Gamma)$ of an entity e and a partial temporal order $O_t = (I_e, \preccurlyeq'_{A_1}, \ldots, \preccurlyeq'_{A_n})$.

We say that O_t is *implied by* S_e, denoted by $S_e \models O_t$, if and only if *for all* valid completions I^c_t of S_e, $O_t \subseteq I^c_t$. Here $O_t \subseteq I^c_t$ if $\preccurlyeq'_{A_i} \subseteq \preccurlyeq^c_{A_i}$ for all $i \in [1, n]$, where $I^c_t = (I_e, \preccurlyeq^c_{A_1}, \ldots, \preccurlyeq^c_{A_n})$.

The *implication problem* for conflict resolution is to decide, given a valid specification S_e and a partial temporal order O_t, whether $S_e \models O_t$.

That is, no matter how we complete the temporal instance I_t of S_e, as long as the completion is valid, the completion includes O_t in its currency orders. The implication analysis is conducted at step (2) of the framework of Fig. 5, for deducing true values of attributes.

Unfortunately, this problem is also intractable.

THEOREM 4.2. *The implication problem for conflict resolution is coNP-complete. It remains coNP-hard for valid specifications $S_e = (I_t, \Sigma, \Gamma)$ of an entity when (1) both Σ and Γ are fixed; (2) $\Gamma = \emptyset$; or (3) $\Sigma = \emptyset$.*

Proof: The coNP upper bound is verified by providing an NP algorithm for the complement problem. In a nutshell, given a specification $S_e = (I_t, \Sigma, \Gamma)$ and a partial temporal order O_t, the algorithm simply guesses a completion I_t^c of I_t and then verifies whether (i) $I_t^c \models \Sigma$; (ii) $I_t^c \models \Gamma$; and (iii) $O_t \not\subseteq I_t^c$. If I_t^c passes these checks successfully, then the algorithm returns "yes" since $S_e \not\models O_t$. Otherwise, the current guess is rejected. This is clearly an NP algorithm for the complement problem and hence the implication problem is in coNP.

For the lower bounds, we show that the implication problem is coNP-hard when (1) both Σ and Γ are fixed; (2) $\Gamma = \emptyset$; or (3) $\Sigma = \emptyset$. The lower bounds for (1) and (2) are established by a revision of the proof of Theorem 4.1. More specifically, we revise the reduction used there as follows. First, the relation schema used in that proof is extended with an additional attribute A. Second, each tuple t in the temporal instance I_t has now two copies: a tuple t^a with its A-attribute set to a constant a, *i.e.,* $t^a[A] = a$, and a tuple t^b with its $t^b[A] = b$. Finally, the premise of each currency constraint used in that proof carries an additional condition "$t_1[A] = a \wedge t_2[A] = b \wedge t_1[A] \preccurlyeq_A t_2[A]$". These conditions enforce the constraints to have an effect only on completions in which b is more current than a in attribute A.

Denote by $S'_e = (I'_t, \Sigma', \Gamma = \emptyset)$ the specification obtained from S_e in the proof of Theorem 4.1 after such revisions. Let O_t be the partial temporal order $(I'_t, \{t^b \preccurlyeq_A t^a\}, \emptyset, \ldots, \emptyset)$, where t^a and t^b are the two copies of an arbitrary tuple t in I_t. We claim the following: (i) S'_e is valid; and (ii) $S'_e \models O_t$ if and only if the formula φ is not satisfiable. For (i) it suffices to observe that for any completion $(I'_t)^c$, as long as it puts $t^b \preccurlyeq_A t^a$ in its currency order for A and arbitrarily completes currency orders for all the other attributes, it makes a valid completion. Indeed, this is simply because the conditions added to the premise of constraints used in the proof of Theorem 4.1 are false, and hence the currency constraint vacuously hold. Hence, S'_e is valid.

For (ii), assume first that there exists a truth assignment μ_X that makes φ true. We define a completion $(I'_t)^c$ of I'_t by setting $t^a \preccurlyeq_A t^b$, where t is the tuple used to define O_t, and by completing the currency orders for the attributes based on μ_X as in the proof of Theorem 4.1. As a result, $O_t \not\subseteq (I'_t)^c$ and $S'_e \not\models O_t$. Conversely, suppose that $S'_e \not\models O_t$. This implies the existence of a valid completion $(I'_t)^c$ of I'_t that includes $t^a \preccurlyeq_A t^b$ and satisfies all currency constraints in Σ'. Similar to the proof of Theorem 4.1 it is readily verified that a truth assignment μ_X can be constructed from $(I'_t)^c$ that makes φ true. Hence, $S'_e \models O_t$ if and only if φ is not satisfiable. Observe that the proof only uses a fixed set of currency constraints and does not require any constant CFDs.

Similarly, the coNP-lower bound for (3) is established by a similar modification of the specification for its counterpart given in the proof of Theorem 4.2, by reduction from the tautology problem. More specifically, given an instance φ of the tautology

problem as stated in the proof of Theorem 4.1, we extend the schema R' given there with an additional attribute A. Its temporal instance I'_t now consists of two tuples $t_0 = (a, 0, \ldots, 0)$ and $t_1 = (b, 1, \ldots, 1)$. We further extend the premises of the constant CFDs ψ_j and ψ_C in the proof of Theorem 4.1 with the extra condition "$t[A] = a$". That is, these constant CFDs only have an effect when the current tuple has a as its A-attribute value. Denote by $S'_e = (I'_t, \Sigma = \emptyset, \Gamma')$ the specification obtained in this way. Clearly, S'_e is consistent since we just need to enforce $t_0 \preccurlyeq^c_A t_1$ in a completion to assure that the corresponding current tuple vacuously satisfies the CFDs in Γ'. Consider $O_t = (I'_t, \{t_0 \preccurlyeq^c_A t_1\}, \emptyset, \ldots, \emptyset)$. Then, similar to the argument given above, one can readily verify that $S'_e \models O_t$ if and only if φ is a tautology. □

True value deduction. The third problem is the *true value problem* for conflict resolution. It is to decide, given a valid specification S_e for an entity e, whether $\mathsf{T}(S_e)$ exists. That is, there exists a tuple t_c such that for all valid completions I^c_t of S_e, $\mathsf{LST}(I^c_t) = t_c$.

This analysis is needed by step (3) of the framework (Fig. 5) to decide whether S_e has enough information to deduce $\mathsf{T}(S_e)$, *i.e.,* whether additional temporal information is needed to determine the true value of e.

No matter how important this problem is, it is also nontrivial: it is coNP-complete, and remains intractable in several practical special cases.

THEOREM 4.3. *The true value problem for conflict resolution is coNP-complete. It remains coNP-hard for valid specifications $S_e = (I_t, \Sigma, \Gamma)$ for an entity even when (1) both Σ and Γ are fixed; (2) $\Gamma = \emptyset$; or (3) $\Sigma = \emptyset$.*

Proof: The upper bound is verified by providing an NP algorithm for the complement problem. Given a specification $S_e = (I_t, \Sigma, \Gamma)$, the algorithm simply guesses *two* completions I^c_t and $(I^c_t)'$ of I_t and then checks whether both completions are valid and generate *different* current tuples. If so, the algorithm returns "yes" and concludes that no true value of S_e can be determined. Otherwise, the current guesses are rejected. This is clearly an NP algorithm for the complement problem, and hence the true value problem is in coNP.

For the lower bounds, we need to show that the true value problem is coNP-hard when (1) both Σ and Γ are fixed; (2) $\Gamma = \emptyset$; or (3) $\Sigma = \emptyset$. The lower bounds for (1) and (2) are verified by a modification of the proof of its counterpart for Theorem 4.2. Indeed, it suffices to add two tuples $t^a_\# = (a, \#, \ldots, \#)$ and $t^b_\# = (b, \#, \ldots, \#)$ to the temporal instance given there, together with additional currency constraints that enforce $\#$ to come after any other constant in the currency orders for all attributes of the schema except for A (which does not carry $\#$). Denote by $S''_e = (I''_t, \Sigma'', \Gamma = \emptyset)$ the specification obtained in this way from S'_e given in the proof of Theorem 4.2. As a consequence, any completion of S''_e can only yield current tuples $t^a_\#$ or $t^b_\#$.

As argued there, S''_e is valid since one only has to consider a completion that includes $t^b_\# \preccurlyeq_A t^a_\#$. Furthermore, we next show that a true value exists if and only if φ is not satisfiable. Indeed, suppose that φ is not satisfiable. Then for any valid completion $(I''_t)^c$ of I''_t, if $(I''_t)^c \models \Sigma''$, then it has to set $t^b_\# \preccurlyeq_A t^a_\#$, since otherwise the currency constraints will be triggered and the completion would generate a satisfying truth assignment for φ, which by assumption does not exist. Hence, the true value will be the tuple $t^a_\#$. Conversely, suppose that no true value exists. This implies that there exist two completions of I''_t, such that one leads to current tuple $t^a_\#$, and the other one leads to current tuple $t^b_\#$. In the second case, $t^a_\# \preccurlyeq_A t^b_\#$ and hence, as argued in the

proof of Theorem 4.2, one can construct a satisfying truth assignment for φ from the completions. Hence, if no true value exists, then φ must be satisfiable.

The coNP-lower bound for (3) is established by a modification of the specification given in the proof for the case of constant CFDs in Theorem 4.2, by reduction from the tautology problem. The modification is as follows. Given an instance φ of the tautology problem as stated in the proof of Theorem 4.2, we introduce a third tuple $t_b = (b, b, \ldots, b)$ to the temporal instance given there, and extend the set Γ' of constant CFDs by including $\psi^i_{a\neg b} = \forall t\ (t[A] = a \wedge t[X_i] = b \rightarrow t[A] = b)$, for $i \in [1, n]$. These constant CFDs prevent the current tuple t in completions from having $t[A] = a$ and $t[X_i] = b$ for all $i \in [1, n]$. In addition, we add $\psi^i_{bb} = \forall t\ (t[A] = b \rightarrow t[X_i] = b)$ and $\psi_{bb} = \forall t\ (t[A] = b \rightarrow t[C] = c)$. These assure that for all current tuples t, it $t[A] = b$, then t has the constant b in all of its attributes.

Denote by $S''_e = (I''_t, \Sigma = \emptyset, \Gamma'')$ the specification obtained this way from S'_e given in the proof of Theorem 4.2. A completion that results in current tuple t_b is clearly a valid completion, and hence S''_e is valid itself. Moreover, it is readily verified that a true value exists if and only if φ is a tautology. Indeed, observe first that completions either result in the current tuple t_b or a tuple of the form $(a, \mu_X, 0)$, where μ_X is a truth assignment for X. While t_b can always be witnessed by a valid completion of S''_e (as mentioned above), $(a, \mu_X, 0)$ can only be witnessed provided that μ_X makes φ false (using the argument given in the proof of Theorem 4.2). Hence t_b is the true value if and only if φ is a tautology. □

Coverage analysis. Finally, the *minimum coverage problem* is to determine, given a valid specification $S_e = (I_t, \Sigma, \Gamma)$ of an entity and a positive integer k, whether there exists a partial temporal order O_t such that (1) $\mathsf{T}(S_e \oplus O_t)$ exists, and (2) $|O_t| \leq k$.

Intuitively, this is to check whether one can add a partial temporal order O_t of a *bounded* size to a specification such that the enriched specification has sufficient information to deduce all the true values of an entity. The ability to solve this problem helps us identify what *minimum* additional temporal information is needed to deduce the true value. The analysis of *minimum* O_t is required by step (4) of the framework of Fig. 5.

This problem is Σ^p_2-complete ($\mathrm{NP}^{\mathrm{NP}}$ or $\mathrm{NP}^{\mathrm{coNP}}$), unfortunately. Worse still, it remains Σ^p_2-hard even in several practical special cases, as stated below.

THEOREM 4.4. *The minimum coverage problem is Σ^p_2-complete. It remains Σ^p_2-hard for valid specifications $S_e = (I_t, \Sigma, \Gamma)$ for an entity even when (1) both Σ and Γ are fixed; (2) $\Gamma = \emptyset$; or (3) $\Sigma = \emptyset$.*

Proof: For the Σ^p_2 upper bound it suffices to observe that the following $\mathrm{NP}^{\mathrm{coNP}}$ algorithm correctly decides whether there exists a partial temporal order O_t of size $|O_t| \leq k$ such that $\mathsf{T}(S_e \oplus O_t)$ exists. Given a valid specification $S_e = (I_t, \Sigma, \Gamma)$, the algorithm first guesses a partial temporal order O_t and then checks whether $|O_t| \leq k$ and whether $\mathsf{T}(S_e \oplus O_t)$ exists. The latter can be done in coNP (see Theorem 4.3). If the guessed partial temporal order passes these checks, then the algorithm returns "yes". Otherwise, the guessed order is rejected. The algorithm is in Σ^p_2 since it is a non-deterministic PTIME algorithm by calling a coNP oracle (see, *e.g.,* [Papadimitriou 1994] for detailed discussion about Σ^p_2).

We now show that the problem is Σ^p_2-hard when (1) Σ and Γ are fixed; (2) $\Gamma = \emptyset$; or (3) $\Sigma = \emptyset$.

For (1) and (2) we establish the Σ^p_2-lower bound by reduction from the $\exists^*\forall^*\mathsf{DNF}$ problem, which is known to be Σ^p_2-complete [Stockmeyer 1976]. An instance of the

$\exists^*\forall^*$DNF problem is a formula of the form $\varphi = \exists X \forall Y \psi$, where $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$, $\psi = C_1 \vee \cdots \vee C_r$; for $j \in [1, r]$, $C_j = \ell_1^j \wedge \ell_2^j \wedge \ell_3^j$, and for $k = 1, 2, 3$, the literal ℓ_k^j is either a variable or the complement of a variable in $X \cup Y$. It is to determine whether φ is true.

Given an instance φ of the $\exists^*\forall^*$DNF problem, we define a specification $S_e = (I_t, \Sigma, \Gamma)$ and a constant k such that the minimal coverage problem for S_e and k has a solution if and only if φ is true. In particular, in S_e we have a fixed set of currency constraints and no constant CFDs. Hence, the reduction shows (1) and (2).

Recall the relation schema $R(A, D, C, P, U, V, W)$ used in the proof of Theorem 4.2. We populate its temporal instance $I_t = (I, \preccurlyeq_A, \preccurlyeq_D, \preccurlyeq_C, \preccurlyeq_P, \preccurlyeq_U, \preccurlyeq_V, \preccurlyeq_W)$ as follows. We assume the presence of $2(n + m)$ distinct constants a_i and b_i for $i \in [1, n]$ and c_i and d_i for $i \in [1, m]$. As in the proof of Theorem 4.2, truth values for variables in X are encoded by means of two tuples:

$$(a, 0, x_i, 0, a_i, a_i, a_i) \quad \text{and} \quad (a, 0, x_i, 0, b_i, b_i, b_i).$$

with their A-attribute set to a, and two tuples

$$(b, 0, x_i, 0, a_i, a_i, a_i) \quad \text{and} \quad (b, 0, x_i, 0, b_i, b_i, b_i).$$

with their A-attribute set to b. Similarly, truth values for variables in Y are encoded by the following tuples:

$$\begin{array}{ccc} (a, 0, y_i, 0, c_i, c_i, c_i) & & (a, 0, y_i, 0, c_i, c_i, c_i) \\ (b, 0, y_i, 0, d_i, d_i, d_i) & \text{and} & (b, 0, y_i, 0, d_i, d_i, d_i). \end{array}$$

Moreover, we add a currency constraint to Σ for every pair of attributes (L, L') taken from $\{U, V, W\}$:

$$\forall t_1, t_2 \; (t_1[D] = 0 \wedge t_2[D] = 0 \wedge t_1[C] = t_2[C] \wedge t_1[A] = a \wedge t_2[A] = b \\ \wedge t_1[A] \prec_A t_2[A] \wedge t_1[L] \prec t_2[L] \rightarrow t_1[L'] \prec t_2[L']).$$

These constraints ensure that whenever $a \prec_A b$, the order between a_i and b_i (resp. c_i and d_i) is consistent for all attributes U, V and W. As before, $a_i \prec_U b_i$ indicates that x_i is set to true, whereas $b_i \prec_U a_i$ indicates that x_i is false; similarly for variables in Y but using the constants c_i and d_i instead. In other words, with every completion of I_t in which $a \prec_A b$, we can associate truth assignments μ_X and μ_Y of X and Y, respectively.

We next encode the clauses in φ in a similar way the one given in the proof of Theorem 4.2. More specifically, given $C_1 \vee \cdots \vee C_r$ we encode its negation $\bar{C}_1 \wedge \cdots \wedge \bar{C}_r$ with $\bar{C}_j = \bar{\ell}_1^j \vee \bar{\ell}_2^j \vee \bar{\ell}_3^j$. Such clauses can be equivalently written as $\ell_1^j \wedge \ell_2^j \rightarrow \bar{\ell}_3^j$ by means of the tuples

$$(a, 1, j, 1, v_1, v_2, v_3) \quad \text{and} \quad (a, 1, j, 2, v_1', v_2', v_3'),$$

and their b-variants

$$(b, 1, j, 1, v_1, v_2, v_3) \quad \text{and} \quad (a, 1, j, 2, v_1', v_2', v_3').$$

Here $v_i = a_k$ and $v_i' = b_k$ if $\ell_i^j = x_k$, and $v_i = b_k$ and $v_i' = a_k$ if $\bar{\ell}_i^j = x_k$, for $i = 1, 2$; we define v_i and v_i' the other way around for $i = 3$; similarly for variables in Y but then using constants c_i and d_i instead. For example, consider the clause $C = x_1 \wedge \bar{x}_2 \wedge \bar{y}_3$ whose complement is $\bar{C} = \bar{x}_1 \vee x_2 \vee y_3$. Equivalently, we write $\bar{C}$ as $x_1 \wedge \bar{x}_2 \rightarrow y_3$. Hence, we encode $\bar{C}$ by $(a, 1, \_, 1, a_1, b_2, c_3)$ and $(a, 1, \_, 2, b_1, a_2, d_3)$, together with their b-counterparts $(b, 1, \_, 1, a_1, b_2, c_3)$ and $(b, 1, \_, 2, b_1, a_2, d_3)$.

The link between truth assignments selected by completions and the validity of (complemented) clauses is established by the following currency constraint:

$$\forall t_1, t_2\ (t_1[D] = 1 \wedge t_2[D] = 1 \wedge t_1[C] = t_2[C] \wedge t_1[P] = 1 \wedge t_1[P] = 2 \wedge t_1[A] = a \wedge t_2[A] = b \\ \wedge\ t_1[A] \prec_A t_2[A] \wedge t_1[U] \prec t_2[U] \wedge t_1[V] \prec t_2[V] \rightarrow t_1[W] \prec t_2[W].$$

This constraint tells us that whenever the truth assignment (represented by a completion) makes $\ell_1^j \wedge \ell_2^j$ true, then it must also make $\bar{\ell}_3^j$ true, provided that $a \prec_A b$.

We also include two tuples $t_\#^a = (a, \#, \ldots, \#)$ and $t_\#^b = (b, \#, \ldots, \#)$ in I_t, which serves as potential true values of the entity represented by S_e. We enforce the symbol $\#$ to come after any other constant in currency orders by means of currency constraints (one for each attribute in R), as defined in the proof of Theorem 4.3. Clearly, in valid completions, if $a \prec_A b$ then $t_\#^b$ is the current tuple; when $b \prec_A a$, $t_\#^a$ is the current tuple.

Finally, we ensure that the partial temporal order O_t can only add currency information related to the values a_i and b_i in the instance, so that O_t can only affect the choice of truth values for variables in X. To achieve this, observe that given instance I_t constructed so far, $|O_t|$ can be no larger than $7|I|^2$, where 7 is simply the number of attributes in R. We let $k = 7|I|^2$. Next, for each constant v different from the a_i's and b_i's we add $p > k$ tuples of the form $(v_{\text{id}}, v, \ldots, v)$, where v_{id} is a unique identifier for each of these tuples. Let I_t' denote the temporal instance obtained in this way and let $S_e = (I', \Sigma, \Gamma = \emptyset)$. Clearly, for any O_t that relates tuples in an attribute with values different from a_i and b_i, O_t will cause the addition of more than k tuples. Indeed, let B be an arbitrary attribute. Then the addition of $t \prec_B t'$ implies that $s \prec_B s'$ for all tuples s and s' that share the same B-attribute value with t and t', respectively. By the choice of k and the addition of $p > k$ tuples for each constant, any O_t of size $\leq k$ can only relate tuples that contain a_i or b_i values in one of its attributes.

Observe that the specification S_e defined above is valid. Indeed, any completion that makes a more current than b in the A-attribute vacuously satisfies the currency constraints in Σ. As a consequence $t_\#^a$ will always be one of the possible current tuples.

We next show that the minimum coverage problem has a solution if and only if φ is true. Suppose first that φ is false. In other words, for every μ_X of X, there exists a truth assignment μ_Y of Y that makes $C_1 \vee \cdots \vee C_r$ false. Consider a partial temporal order O_t with $|O_t| \leq k$. By the construction, O_t can only add temporal information between tuples that concern variables in X. In other words, the impact of O_t is that it restricts the set of truth assignments of X that can be obtained by means of valid completions. However, since φ is false, even for each μ_X in this restricted set, there exists a μ_Y that makes the $C_1 \vee \cdots \vee C_r$ false. This in turn implies that $t_\#^b$ can be a current tuple in a completion that sets $a \prec_A b$. Indeed, simply consider the completion that (i) sets $a \prec_A b$; (ii) selects a μ_X that belongs to the restricted set; (iii) selects μ_Y such that the clauses are false; and (iv) arbitrarily completes partial currency orders for the other attributes. It is easily verified that this completion indeed satisfies all currency constraints since it satisfies the constraints related to truth assignments and all constraints corresponding to the negated clauses (recall that μ_X and μ_Y make all $\bar{C}_j$ true). Hence, when φ is false, both $t_\#^a$ and $t_\#^b$ are current tuples and no true value can exist, no matter what O_t is.

Conversely, suppose that φ is true. That is, there exists a truth assignment μ_X of X such that for all μ_Y of Y, $C_1 \vee \cdots \vee C_r$ is true. We let O_t be the partial temporal order that restricts the choices of truth assignments for X to be μ_X. By the construction, this can be done by using $\leq k$ added pairs. Then it is impossible that $t_\#^b$ becomes a current tuple. Indeed, for this to happen we need a completion that sets $a \prec_A b$ and

	satisfiability	implication	true value	minimum coverage
complexity	NP-complete	coNP-complete	coNP-complete	Σ_2^p-complete
(special cases) both Σ and Γ are fixed	NP-complete	coNP-complete	coNP-complete	Σ_2^p-complete
only currency constraints are present	NP-complete	coNP-complete	coNP-complete	Σ_2^p-complete
only constant CFDs are present	NP-complete	coNP-complete	coNP-complete	Σ_2^p-complete

Fig. 6. Complexity of reasoning about conflict resolution

in addition, satisfies all constraints in Σ. This, however, would imply the existence of a truth assignment μ_Y of Y, which, together with μ_X, makes $C_1 \vee \cdots \vee C_r$ false. This is contrast to the assumption that φ holds for μ_X. As a consequence, $\mathsf{T}(S_e \oplus O_t)$ exists and is equal to $t^a_{\#}$.

Finally, we show that the problem is Σ_2^p-hard for case (3), when $\Sigma = \emptyset$. This is verified again by reduction from the $\exists^*\forall^*$DNF problem, but now we use constant CFDs only. The idea behind the reduction is similar to that of the reduction given for cases (1) and (2).

Given an instance φ of the $\exists^*\forall^*$DNF problem, we define a relation schema $R(A, X_1, \ldots, X_n, Y_1, \ldots, Y_m, C)$. To populate its corresponding temporal instance I_t, we start with two tuples $t_0 = (a, 0, 0, \ldots, 0)$ and $t_1 = (b, 1, 1, \ldots, 1)$. Completions thus lead to current tuples ranging over all possible truth assignments for X and Y. We further introduce a tuple $t_b = (b, b, \ldots, b)$, which will correspond to the true value of the entity if it exists. Finally, let $k = n$ and add $p > n$ tuples of the form $(c_i, \ldots, c_i, 0, 0, 0, \ldots)$ and $(c_i, \ldots, c_i, 1, \ldots, 1)$ to I_t, for $i \in [1, p]$. Here the c_i's are values of the attributes A and $X_1, \ldots, X_n$. We further assume that initial temporal orders are available, asserting that the c_i's come before a, b, 0 and 1. Intuitively, the addition of these p tuples will cause any additional temporal information in the Y-attributes (and A-attribute) to have more than "k effects", *i.e.,* if $t_0 \prec_Y t_1$ is in O_t, then this addition needs to be imposed on all p tuples as well since these tuples contain the same values in their Y-attributes as t_0 and t_1. As a consequence, any partial temporal order O_t of size $\leq k$ can only enrich currency orders for the X-attributes. In other words, adding O_t will cause the selection of a truth assignment for X.

We use the same constant CFDs as those defined in the proof of Theorem 4.3, and let $S_e = (I_t, \Sigma = \emptyset, \Gamma)$ be the resulting specification. As argued in the proof of Theorem 4.3, S_e is valid because any completion of S_e with $a \prec_A b$ satisfies the CFDs in Γ. Recall also that t_b will be the current tuple in this case.

We next show that the minimum coverage problem has a solution for S_e and k if and only if φ is true. Indeed, suppose that φ is false. Then for all truth assignments μ_X of X, there exists a μ_Y of Y, such that $C_1 \vee \cdots \vee C_r$ is false. Let O_t be any partial temporal order of size $\leq k$. As argued above, the addition of O_t causes the selection of a subset of truth assignments of X. For any such μ_X we have a μ_Y that makes the clauses false. In other words, a completion exists, which (i) puts $b \prec_A a$; (ii) selects a μ_X; and (iii) picks μ_Y that falsifies φ. By the definition of the CFDs, this implies that a current tuple of the form $(a, \ldots)$ exists and hence there is no true value for the entity (since t_b is also a current tuple).

Conversely, if φ is true, we simple take O_t that selects the satisfying truth assignment μ_X of X such that for all μ_Y of Y, all the clauses in φ are satisfied. Such O_t can be taken of size $\leq k$. In other words, completions with $b \prec_A a$ cannot exist by the definition of the constant CFDs. Hence, t_b is the only possible current tuple and thus the true value of S_e exists. □

Remark. The complexity results are summarized in Fig. 6. From these results we find the following.

(i) The main conclusion is that while these problems are important in practice, they are

hard. In fact, we have shown that the lower bounds of all these problems remain intact for specifications $S_e = (I_t, \Sigma, \Gamma)$ of an entity when (1) both Σ and Γ are fixed; (2) $\Gamma = \emptyset$, *i.e.,* when constant CFDs are absent; or (3) $\Sigma = \emptyset$, *i.e.,* when currency constraints are absent. Hence unless P = NP, efficient (PTIME) algorithms for solving these problems are necessarily *heuristic*.

(ii) The results not only reveal the complexity of reasoning about conflict resolution, but also advance our understanding of data currency and consistency. Indeed, while the minimum coverage problem is specific for conflict resolution and has *not* been studied before, the other three problems are also of interest to the study of data currency. Taken together with the complexity results of [Fan et al. 2012], Theorems 4.1, 4.2 and 4.3 show that currency constraints make our lives easier as opposed to denial constraints: they reduce the complexity of inferring data currency [Fan et al. 2012], from Σ_2^p-complete, Π_2^p-complete (coNP$^{\text{NP}}$) and Π_2^p-complete down to NP-complete, coNP-complete and coNP-complete, respectively,

When it comes to data consistency, it is known that the satisfiability and implication problems for general CFDs are NP-complete and coNP-complete, respectively [Fan et al. 2008]. Theorems 4.1 and 4.2 give a *stronger* result: these lower bounds already hold for constant CFDs.

5. ALGORITHMS FOR CONFLICT RESOLUTION

We next provide algorithms underlying the framework depicted in Fig. 5. We first present an algorithm for checking whether a specification is valid (step (1) of the framework; Section 5.1). We then study how to deduce true attribute values from a valid specification (step (2); Section 5.2). Since not all true attribute values can be deduced automatically, we further discuss algorithms to generate suggestions such that the users may provide true values of some attributes (step (4); Section 5.3), which can in turn help the deduction procedure.

5.1. Validity Checking

We start with algorithm IsValid that, given a specification $S_e = (I_t, \Sigma, \Gamma)$, returns true if S_e is valid, and false otherwise. As depicted in Fig. 5, IsValid is invoked for an initial specification S_e and its extensions $S_e \oplus O_t$ with the input O_t from the users.

Theorem 4.1 tells us that it is NP-complete to determine whether S_e is valid. In other words, IsValid is necessarily heuristic if it is to be efficient. Instead of designing an efficient algorithm from scratch, we approach this by reducing the problem to SAT, one of the most studied NP-complete problems, which is to decide whether a Boolean formula is satisfiable (see *e.g.,* [Biere et al. 2009]). Several high-performance tools for SAT (SAT-solvers) are already in place [Biere et al. 2009], which have proved effective in *e.g.,* software verification, AI and operations research. For instance, MiniSAT [Giunchiglia and Tacchella 2004] can effectively solve a formula with $4,500$ variables and $100K$ clauses in 1 second.

Algorithm IsValid leverages existing SAT-solvers. We convert a given specification S_e to a propositional formula in the conjunctive normal form (CNF), and then employ a SAT-solver to decide the satisfiability of S_e.

Algorithm. More specifically, given a specification S_e of an entity e, IsValid works in three steps as follows.

(i) Instantiation: First, the specification S_e is expressed as a set of (propositional) predicate formulas.

(ii) ConvertToCNF: Then, the predicate formulas from (i) are converted into a CNF such that the given specification is valid if and only if the CNF is satisfiable.

(iii) Finally, a SAT-solver is applied to the CNF resulting from (ii) to determine the satisfiability of the CNF.

We next present procedures Instantiation and ConvertToCNF. Consider $S_e = (I_t, \Sigma, \Gamma)$, where $I_t = (I_e, \preccurlyeq_{A_1}, \ldots, \preccurlyeq_{A_n})$ is a temporal instance of schema R. We denote also by R the set $\{A_i \mid i \in [1, n]\}$ of attributes in R. We define the *extended active domain* of A_i, denoted by $\mathsf{adom}^v(I_e.A_i)$, to be the set including all the values in $\mathsf{adom}(I_e.A_i)$ and all the constants that appear in attribute A_i of some constant CFD in Γ. To check whether S_e is satisfiable, it suffices to consider the values from the extended active domains only.

Instantiation. To uniformly treat partial currency orders, currency constraints and constant CFDs as predicate formulas, we introduce a notion of *instance constraints*. The set of instance constraints of S_e, denoted as $\Omega(S_e)$, is defined in terms of values in the extended active domains, and a strict partial order $\prec^v_{A_i}$ on $\mathsf{adom}^v(I_e.A_i)$. These constraints are derived from S_e as follows.

(1) *Currency orders*. To encode the partial currency orders in I_t, for each $A_i \in R$, we include the following instance constraints in $\Omega(S_e)$.

(a) Partial orders in I_t: $(\mathit{true} \rightarrow t_1[A_i] \prec^v_{A_i} t_2[A_i])$ for each $t_1 \preccurlyeq_{A_i} t_2$ in I_t, as long as $t_1[A_i] \neq t_2[A_i]$.

(b) Transitivity of $\prec_{A_i}$: $(a_1 \prec^v_A a_2 \wedge a_2 \prec^v_A a_3 \rightarrow a_1 \prec^v_A a_3)$ for all distinct values a_1, a_2, a_3 in $\mathsf{adom}^v(I_e.A_i)$.

(c) Asymmetry: $(a \prec^v_{A_i} b) \rightarrow \neg(b \prec^v_{A_i} a)$ for all distinct values $a, b \in \mathsf{adom}^v(I_e.A_i)$.

Intuitively, these assure that each $\prec^v_{A_i}$ is a *strict partial order* (via both (b) and (c)), and express available temporal information in I_t as predicate formulas (via (a)).

(2) *Currency constraints*. For each currency constraint $\varphi = \forall t_1, t_2\ (\omega \rightarrow t_1 \prec_{A_r} t_2)$ in Σ and for all distinct tuples $s_1, s_2 \in I_e$, we include the following in $\Omega(S_e)$:

$$\mathsf{ins}(\omega, s_1, s_2) \ \rightarrow\ s_1[A_r] \prec^v_{A_r} s_2[A_r],$$

where $\mathsf{ins}(\omega, s_1, s_2)$ is obtained from ω by (a) substituting $s_i[A_j]$ for t_i and $\prec^v_{A_j}$ for $\prec_{A_j}$ in each predicate $t_1 \prec_{A_j} t_2$, for $i \in [1, 2]$; and (b) evaluating each conjunct of ω defined with a comparison operator to its truth value *w.r.t.* s_1 and s_2. Intuitively, $\mathsf{ins}(\omega, s_1, s_2)$ "instantiates" ω with values in s_1 and s_2.

Example 7: For currency constraint φ_1 in Fig. 3, and tuples r_1 and r_2 in Fig. 2 for Edith, its instance constraint is derived to be ($\mathit{true} \rightarrow \mathit{working} \prec^v_{\mathsf{status}} \mathit{retired}$). Observe that the precondition of φ_1 is evaluated to be true on these two particular tuples, by instantiating variables of φ_1 with values in r_1 and r_2.

Similarly, from currency constraint φ_6 and tuples r_1 and r_2, we derive instance constraint ($\mathit{working} \prec^v_{\mathsf{status}} \mathit{retired} \rightarrow \mathit{212} \prec^v_{\mathsf{AC}} \mathit{415}$), by replacing $\prec_{\mathsf{status}}$ with $\prec^v_{\mathsf{status}}$, and by replacing variables in φ_6 with the corresponding attribute values from r_1 and r_2. □

(3) *Constant CFDs*. For each CFD $\forall t\ (\nu \rightarrow t[A_r] = c_r)$ in Γ and each $c \in \mathsf{adom}^v(I_e.A_r) \backslash \{c_r\}$, $\Omega(S_e)$ includes

$$\psi = (\nu' \ \rightarrow\ c \prec^v_{A_r} c_r),$$

where ν' is obtained from ν by replacing each conjunct $t[A_j] = c_j$ in ν with the conjunction $\bigwedge_c c \prec^v_{A_j} c_j$, where c ranges over all the values in $\mathsf{adom}^v(I_e.A_j) \setminus \{c_j\}$.

Intuitively, constraint ψ asserts that if ν describes true values in the attributes in ν, then c_r is the true value of attribute A_r. Indeed, the CFD is defined on $\mathsf{LST}(I^c_t)$ for a completion I^c_t of I_t (see Section 2.2), and ψ assures that this semantics is enforced.

Example 8: Recall constant CFD ψ_1 from Fig. 3. For the entity instance E_1 of Edith, the CFD is encoded by two instance constraints given below, included in Ω_{E_1}:

$$\textit{212} \prec^v_{\mathsf{AC}} \textit{213} \wedge \textit{415} \prec^v_{\mathsf{AC}} \textit{213} \rightarrow NY \prec^v_{\mathsf{city}} LA,$$
$$\textit{212} \prec^v_{\mathsf{AC}} \textit{213} \wedge \textit{415} \prec^v_{\mathsf{AC}} \textit{213} \rightarrow SFC \prec^v_{\mathsf{city}} LA,$$

i.e., LA is her true city value if her true AC is *213*. □

ConvertToCNF. After we derive $\Omega(S_e)$, we convert the instance constraints of $\Omega(S_e)$ into a CNF $\Phi(S_e)$ as follows. We first substitute a Boolean variable $x^{A_i}_{a_1a_2}$ for each predicate $a_1 \prec^v_{A_i} a_2$ in $\Omega(S_e)$. We then rewrite each formula of the form $(x_1 \wedge \cdots \wedge x_k \rightarrow x_{k+1})$ into the equivalent form $(\neg x_1 \vee \cdots \vee \neg x_k \vee x_{k+1})$. Finally, $\Phi(S_e)$ is defined to be the conjunction of all such formulas obtained from $\Omega(S_e)$, which is obviously in CNF.

One can readily verify the following (by contradiction), which justifies the reduction from the satisfiability of the specification S_e to the SAT instance $\Phi(S_e)$.

LEMMA 5.1. *Specification S_e is valid if and only if its converted* CNF $\Phi(S_e)$ *is satisfiable.*

Complexity: Observe the following. (a) The size $|\Omega(S_e)|$ of $\Omega(S_e)$ is bounded by $O((|\Sigma| + |\Gamma|)|I_t|^2 + |I_t|^3)$, since encoding currency orders, currency constraints and constant CFDs is in time $O(|I_t|^3)$, $O(|\Sigma||I_t|^2)$ and $O(|\Gamma||I_t|^2)$, respectively. (b) It takes $O(|\Omega(S_e)|)$ time to convert $\Omega(S_e)$ into $\Phi(S_e)$. Hence the size of the CNF $\Phi(S_e)$ is bounded by $O((|\Sigma| + |\Gamma|)|I_t|^2 + |I_t|^3)$. In practice, an entity instance I_t is typically *much smaller than* a database, and the sets Σ and Γ of constraints are also *small*. As will be seen in Section 6, SAT-solvers can efficiently process CNFs of this size.

5.2. Deducing True Values

We next present an algorithm that, given a valid specification $S_e = (I_t, \Sigma, \Gamma)$ of an entity e, deduces *true values* for as many attributes of e as possible.

Intuitively, it is to find a maximum partial order O_d such that $S_e \models O_d$, *i.e.,* (a) for all valid completions I^c_t of S_e, $O_d \subseteq I^c_t$ (Section 4), and (b) for tuples $t_1, t_2 \in I_e$ and $A_i \in R$, if $S_e \models t_1 \prec_{A_i} t_2$ then $t_1 \prec_{A_i} t_2$ is in O_d.

5.2.1. Partial Order Deduction. To deduce true values, below we first present a heuristic approach, and then discuss an exact algorithm.

A heuristic approach. Given S_e, we want to deduce a maximum partial order O_d such that $S_e \models O_d$. As an immediate corollary of Theorem 4.2, one can show that the decision version of this problem is also coNP-complete, even when either Σ or Γ is fixed or absent. Thus we give a heuristics to strike a balance between its complexity and accuracy. The algorithm is based on the following lemma, which is easy to verify.

LEMMA 5.2. *For the* CNF $\Phi(S_e)$ *converted from a valid specification S_e, and for all tuples t_1, t_2 in S_e such that $t_1[A_i] = a_1$ and $t_2[A_i] = a_2$, $S_e \models t_1 \prec_{A_i} t_2$ if and only if $\Phi(S_e) \rightarrow x^{A_i}_{a_1a_2}$ is a tautology, where $x^{A_i}_{a_1a_2}$ is the variable denoting $a_1 \prec^v_{A_i} a_2$ in $\Phi(S_e)$.*

Observe that the condition $\Phi(S_e) \rightarrow x^{A_i}_{a_1a_2}$ indicates that for any truth assignment μ, if μ satisfies $\Phi(S_e)$, then $\mu(x^{A_i}_{a_1a_2})$ is *true*. That is, the one-literal clause $x^{A_i}_{a_1a_2}$ is implied by $\Phi(S_e)$, which in turn encodes S_e. Based on this observation, our algorithm checks one-literal clauses in $\Phi(S_e)$ one by one, and enriches the known partial order accordingly.

Algorithm. The algorithm for deducing true values, referred to as DeduceOrder, is given in Fig. 7. It first converts a specification S_e to the CNF $\Phi(S_e)$ (lines 1-2; see Sec-

Algorithm DeduceOrder
Input: A valid specification $S_e = (I_t, \Sigma, \Gamma)$ of an entity.
Output: A partial temporal order O_d such that $S_e \models O_d$.
1. $\Omega(S_e) :=$ Instantiation(S_e);
2. $\Phi(S_e) :=$ ConvertToCNF$(\Omega(S_e))$;
3. $O_d := (I_e, \emptyset, \ldots, \emptyset)$;
4. **while** there exists a one-literal clause C in $\Phi(S_e)$ **do**
   /* $x^A_{a_1 a_2}$ in C is the variable denoting $a_1 \prec^v_A a_2$ */
5. **if** C is a one-literal clause ($x^A_{a_1 a_2}$) **then**
6. add $a_1 \prec^v_A a_2$ to O_d;
7. $C_\neg := \neg x^A_{a_1 a_2}$;
8. **if** C is a one-literal clause ($\neg x^A_{a_1 a_2}$) **then**
9. add $a_2 \prec^v_A a_1$ to O_d;
10. $C_\neg := x^A_{a_1 a_2}$;
11. **for each** $C' \in \Phi(S_e)$ **do**
12. **if** C' contains $C_\neg$ **then**
13. $C' := C' \setminus C_\neg$;
14. **if** C' contains C **then**
15. Remove C' from $\Phi(S_e)$;
16. **return** O_d.

Fig. 7. Algorithm DeduceOrder

tion 5.1). For each literal C of the form $x^{A_i}_{a_1 a_2}$ or $\neg x^{A_i}_{a_1 a_2}$, it checks whether C is a clause in (*i.e.,* implied by) $\Phi(S_e)$ (line 4); and if so, it will enrich the partial order O_d (lines 5-10). It then reduces $\Phi(S_e)$ by using C and its negation $C_\neg$ (lines 11-15). That is, for each clause C' that contains C, the entire C' is removed since C' is *true* if C has to be satisfied (lines 12-13). Similarly, for each clause C'' that contains $C_\neg$, $C_\neg$ is removed from C'', as $C_\neg$ has to be *false* (lines 14-15). The deduced partial order O_d is then returned (line 16).

Example 9: Given the entity instance E_2 of Fig. 2 and the constraints of Fig. 3, DeduceOrder finds O_d including: (1) *0* $\prec^v_{\mathsf{kids}}$ *2* by φ_4, (2) *working* $\prec^v_{\mathsf{status}}$ *retired* by φ_1, (3) *sailor* $\prec^v_{\mathsf{job}}$ *veteran*, *401* $\prec^v_{\mathsf{AC}}$ *212* and *02840* $\prec^v_{\mathsf{zip}}$ *12404*, by (2) and φ_5, φ_6 and φ_7, respectively.

Assume additionally that the users assure that the true value of the attribute status is *retired*, i.e., *unemployed* $\prec^v_{\mathsf{status}}$ *retired* is part of the initial currency order. Then DeduceOrder can extend O_d including: (1) *n/a* $\prec^v_{\mathsf{job}}$ *veteran* by φ_5, (2) *312* $\prec^v_{\mathsf{AC}}$ *212* by φ_6 and (3) *60653* $\prec^v_{\mathsf{zip}}$ *12404* by φ_7. Furthermore, from *312* $\prec^v_{\mathsf{AC}}$ *212* and *401* $\prec^v_{\mathsf{AC}}$ *212* and the constant CFD ψ_2, DeduceOrder adds *Newport* $\prec^v_{\mathsf{city}}$ *NY* and *Chicago* $\prec^v_{\mathsf{city}}$ *NY* to O_d. Finally, using the information in O_d related to the attributes city and zip, DeduceOrder adds *Rhode Island* $\prec^v_{\mathsf{county}}$ *Accord* and *Bronzeville* $\prec^v_{\mathsf{county}}$ *Accord* to O_d.

This again illustrates that inferences of *currency constraints* help *consistency* inference and *vice versa*. □

Complexity. (1) It takes $O((|\Sigma| + |\Gamma|)|I_t|^2 + |I_t|^3)$ time to convert S_e into $\Phi(S_e)$ (lines 1-2; see Section 5.1). (2) The *total time* taken by the **while** loop (lines 4-15) is in $O((|\Sigma| + |\Gamma|)|I_t|^2 + |I_t|^3)$. Indeed, we maintain a hash-based index for literals C, in which the key is C and its value is the list of clauses in $\Phi(S_e)$ that contain C or $C_\neg$. In the process, $\Phi(S_e)$ decreases monotonically. Hence in total it takes at most $O(|\Phi(S_e)|)$ time to reduce

Algorithm NaiveDeduce
Input: A valid specification $S_e = (I_t, \Sigma, \Gamma)$ of an entity.
Output: A partial temporal order O'_d such that $S_e \models O'_d$.
1. $\Omega(S_e) :=$ Instantiation(S_e);
2. $\Phi(S_e) :=$ ConvertToCNF$(\Omega(S_e))$;
3. $O'_d := (I_e, \emptyset, \ldots, \emptyset)$;
4. **for each** $A \in R$ **do**
5. **for each** $a_1, a_2 \in \mathsf{adom}^v(A)$ **do**
/* $x^A_{a_1a_2}$ is the variable denoting $a_1 \prec^v_A a_2$ */
6. **if** $\neg$ SAT$(\Phi(S_e) \cup \{\neg x^A_{a_1a_2}\})$ **then**
7. add $a_1 \prec^v_A a_2$ to O'_d;
8. **return** O'_d.

Fig. 8. Algorithm NaiveDeduce

$\Phi(S_e)$ for *all literals*, where $|\Phi(S_e)|$ is bounded by $O((|\Sigma|+|\Gamma|)|I_t|^2+|I_t|^3)$. Putting these together, the algorithm works in $O((|\Sigma| + |\Gamma|)|I_t|^2 + |I_t|^3)$ time.

An exact approach. By Lemma 5.2, one might want to compute a temporal order O'_d consisting of all such variables $x^{A_i}_{a_1a_2}$ that $\Phi(S_e) \wedge \neg x^{A_i}_{a_1a_2}$ is *not* satisfiable. That is, for each variable $x^{A_i}_{a_1a_2}$, we inspect the satisfiability of $\Phi(S_e) \wedge \neg x^{A_i}_{a_1a_2}$ by invoking an SAT-solver.

This approach, referred to as NaiveDeduce, is given in Fig. 8. It first converts specification S_e to the CNF $\Phi(S_e)$ (lines 1-2). For each attribute A (line 4), it then enumerates value pairs (a_1, a_2) in $\mathsf{adom}^v(A)$ (line 5). It examines whether $\Phi(S_e) \rightarrow x^A_{a_1a_2}$ is a tautology by invoking the SAT-solver to check whether $\Phi(S_e) \wedge \neg x^A_{a_1a_2}$ is not satisfiable (line 6). If $\Phi(S_e) \rightarrow x^A_{a_1a_2}$ is a tautology, it adds $a_1 \prec^v_A a_2$ to O'_d (line 7). The procedure returns O'_d when all the possible partial orders are examined (line 8).

NaiveDeduce is an exact algorithm for deducing O'_d provided that the SAT-solver it invokes is an exact algorithm. However, NaiveDeduce calls the SAT-solver $|I_t|^2$ times. As will be seen in Section 6, DeduceOrder finds an O_d with accuracy comparable to O'_d, without incurring the cost of repeatedly calling an SAT-solver.

5.2.2. True Value Deduction. Using the partial temporal order O_d found by DeduceOrder or NaiveDeduce, one can readily deduce true attributes values as follows: a value a_1 is the true value of attribute A_i if for all values $a_2 \in \mathsf{adom}^v(I_e.A_i) \setminus \{a_1\}$, the currency order $a_2 \prec^v_A a_1$ is in O_d. Let $\mathcal{B} \subseteq R$ be the set of attributes for which true values $\mathsf{V}_\mathcal{B}$ can be deduced. For all remaining attributes, *i.e.,* those for which true values cannot be deduced, we put variables as placeholders for their true values that may be inferred after more currency information becomes available.

Example 10: Recall the temporal order O_d from Example 9. Let us first consider O_d before we know that the true value of the attribute status is *retired*. Then, the current tuple of George is of the form (*George*, $x_{\mathsf{status}}, x_{\mathsf{job}}$, *2*, $x_{\mathsf{city}}, x_{\mathsf{AC}}, x_{\mathsf{zip}}, x_{\mathsf{county}}$), with variables. In this case, $\mathcal{B} = \{\mathsf{name}, \mathsf{kids}\}$.

In contrast, when the true value of the attribute status is assumed to be *retired*, then we can derive from O_d that George's current tuple is t_2 = (George, retired, n/a, 2, NY, 212, 12404, Accord). In this case $\mathcal{B}$ consists of all attributes. □

5.3. Generating Suggestions

To identify the true value of the entity e specified by $S_e = (I_t, \Sigma, \Gamma)$, instead of asking the users for input on all those attributes whose true values remain unknown, *i.e.,* for attributes in $R \setminus \mathcal{B}$, we compute a suggestion for a set of attributes $\mathcal{A} \subseteq R$, disjoint from $\mathcal{B}$, such that if the true values for $\mathcal{A}$ are validated, the true value of *the entire* e can be determined, even for attributes in $R \setminus (\mathcal{B} \cup \mathcal{A})$ (see Fig. 5). Below we first formally define suggestions and a notion of derivation rules (Section 5.3.1). We then provide an algorithm for computing suggestions (Section 5.3.2).

5.3.1. Suggestions and Derivation Rules. For each attribute $A_i \in R \setminus \mathcal{B}$, we denote by $\mathsf{V}(A_i)$ the set of *candidate true values for* A_i, *i.e.,* for any candidate $a_1 \in \mathsf{V}(A_i)$, there exists no $a_2 \in \mathsf{adom}^v(I_e.A_i) \setminus \{a_1\}$ such that $a_1 \prec^v_A a_2$ is in O_d. For a set X of attributes, we write $\mathsf{V}(X) = \{\mathsf{V}(A_i) \mid A_i \in X\}$.

Suggestion. A *suggestion* for S_e is a pair $(\mathcal{A}, \mathsf{V}(\mathcal{A}))$, where $\mathcal{A} = (A_1, \ldots, A_m)$ is a set of attributes of R such that $\mathcal{A} \cap \mathcal{B} = \emptyset$ and moreover, (1) there exist values $(a_1, \ldots, a_m)$ such that if $(a_1, \ldots, a_m)$ are validated as the true values of $\mathcal{A}$, then the true value $\mathsf{T}(S_e)$ of S_e exists; and (2) for all possible values $(a'_1, \ldots, a'_m)$ that satisfy condition (1), a'_i is in $\mathsf{V}(A_i)$ for $i \in [1, m]$.

Intuitively, condition (1) says that when the true values of $\mathcal{A}$ are validated, so is $\mathsf{T}(S_e)$. That is, the true values of attributes in $\mathcal{A}' = R \setminus (\mathcal{B} \cup \mathcal{A})$ can be automatically deduced from $\mathsf{V}_{\mathcal{B}}$ and the true values of $\mathcal{A}$. Condition (2) says that $\mathsf{V}(\mathcal{A})$ gives "complete" candidates for the true values of $\mathcal{A}$ from their active domains.

One naturally wants a suggestion to be as "small" as possible, so that it takes the users minimal efforts to validate the true values of $\mathcal{A}$. This motivates us to study the *minimum suggestion problem*, which is to find a suggestion $(\mathcal{A}, \mathsf{V}(\mathcal{A}))$ with the minimum number $|\mathcal{A}|$ of attributes. Unfortunately, this problem is Σ_2^p-complete ($\mathrm{NP}^{\mathrm{NP}}$), which can be verified by reduction from the minimum coverage problem (Theorem 4.4).

COROLLARY 5.3. *The (decision version of) minimum suggestion problem for conflict resolution is Σ_2^p-complete.*

Proof: It suffices to observe that a solution of the minimal suggestion problem of size ℓ relates to a solution of the minimal coverage problem of size $k = \ell |I|^2$. Conversely, one can show that a solution of the minimal coverage problem of size k relates to a solution of the minimal suggestion problem of size $\lceil k/|I|^2 \rceil$. □

In light of the high complexity, we develop an effective heuristic algorithm to compute suggestions. To do this, we examine how true values are inferred by using currency constraints and constant CFDs in a specification S_e, by expressing them as a uniform set of rules.

Derivation rules. A *true-value derivation rule* for S_e has the form $(X, P[X]) \rightarrow (B, b)$, where (1) X is a set of attributes, B is a single attribute, (2) b is a value that is either in $\mathsf{adom}^v(I_e.B)$; and (3) for each $A_i \in X$, $P[A_i]$ is drawn from $\mathsf{adom}^v(I_e.A_i)$. It assures if $P[X]$ is the true value of X, then b is the true value of B.

Derivation rules are computed from instance constraints $\Omega(S_e)$ of S_e, which is illustrated below and will be elaborated in Section 5.3.2.

Example 11: Sample rules for George in Fig. 2 include:

n_1 : ({status}, {*retired*}) → (job, *veteran*)
n_2 : ({status}, {*retired*}) → (AC, *212*)
n_3 : ({status}, {*retired*}) → (zip, *12404*)
n_4 : ({city, zip}, {*NY*, *12404*}) → (county, *Accord*)

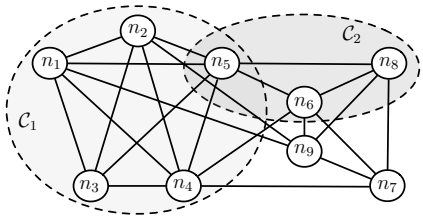


Fig. 9. Sample compatibility graph

n_5 : ({AC}, {*212*}) → (city, *NY*)
n_6 : ({status}, {*unemployed*}) → (job, *n/a*)
n_7 : ({status}, {*unemployed*}) → (AC, *312*)
n_8 : ({status}, {*unemployed*}) → (zip, *60653*)
n_9 : ({city, zip}, {*Chicago*, *60653*}) → (county, *Bronzeville*)

Here rule n_5 is derived from CFD ψ_2, which states that if his true AC is *212*, then his true city must be *NY*. Rule n_1 is from tuple r_5 and constraint φ_5 (Fig. 3), which states that if his true status is *retired*, then his true job is *veteran*. Note that in n_1, status is instantiated with *retired*. Similarly, n_6 is derived from r_6 and φ_5; n_2 and n_3 (resp. n_7 and n_8) are derived from tuple r_5 (resp. r_6) and constraints φ_6 and φ_7, respectively; and n_4 (resp. n_9) is derived from r_5 (resp. r_6) and φ_8. □

The intuition behind our heuristic is as follows. To find a suggestion, we pick a set $\mathcal{A}$ of attributes so that a maximum number of derivation rules can be applied to them, directly or indirectly in the derivation process. As a consequence, we want to derive the true values of as many other attributes as possible from these rules. To capture this idea, we introduce a notion of compatibility graphs, to represent rules that can either be directly applied or be indirectly applied in the derivation process.

Compatibility graphs. Consider a set Π of derivation rules. The *compatibility graph* $G(N, E)$ of Π is an *undirected* graph, where (1) each node x in N is a rule $(X_x, P_x[X_x]) \rightarrow (B_x, b_x)$ in Π, and (2) an edge (x, y) is in E if and only if $B_x \neq B_y$ and $P_x[X_{xy}] = P_y[X_{xy}]$, where $X_{xy} = (X_x \cup B_x) \cap (X_y \cup B_y)$.

Intuitively, two nodes are connected (*i.e.,* compatible) in G if their associated derivation rules derive different attributes (*i.e.,* $B_x \neq B_y$), and they agree on the values of their common attributes (*i.e.,* $P_x[X_{xy}] = P_y[X_{xy}]$). Hence these rules have no conflict with each other and can be applied at the same time.

Example 12: The compatibility graph of the rules given in Example 11 is shown in Fig. 9. There is an edge (n_1, n_2) in the graph since their common attribute status has the same value *retired*; similarly for the other edges. In contrast, there is no edge between n_5 and n_7 since the values of their common attribute AC are different: *212* for n_5 and *312* for n_7. □

Observe that each clique $\mathcal{C}$ in the compatibility graph indicates a set of derivation rules that can be applied together. Let $\mathcal{A}'$ be the set of attributes whose true values can be derived from the rules in $\mathcal{C}$, if $\mathcal{C}$ and S_e have no conflicts (will be discussed shortly). To find a suggestion, we compute a maximum clique $\mathcal{C}$ from the graph, and derive a suggestion as $(\mathcal{A}, \mathsf{V}(\mathcal{A}))$ from $\mathcal{C}$, where $\mathcal{A}$ consists of attributes in $R \setminus (\mathcal{A}' \cup \mathcal{B})$, and $\mathsf{V}(\mathcal{A})$ is the set of candidate true values for $\mathcal{A}$.

Example 13: Example 6 shows that for George with entity instance E_2, only the true values of name and kids are known, *i.e.,* $\mathcal{B} = \{\mathsf{name}, \mathsf{kids}\}$ and $\mathsf{V}_{\mathcal{B}} = (\textit{George, 2})$. To find a suggestion for George, we identify a clique $\mathcal{C}_1$ with five nodes n_1–n_5 in the compatibility

Algorithm Suggest
Input: A specification $S_e = (I_t, \Sigma, \Gamma)$ of an entity,
order O_d ($S_e \models O_d$), and $\mathsf{V}_\mathcal{B}$.
Output: A suggestion $(\mathcal{A}, \mathsf{V}(\mathcal{A}))$.
1. $\mathsf{V}(R) := \mathsf{DeriveVR}(I_t, O_d)$;
2. $\Omega(S_e) := \mathsf{Instantiation}(S_e)$;
3. $\Pi := \mathsf{TrueDer}(\Omega(S_e), \mathsf{V}(R))$;
4. $G := \mathsf{CompGraph}(\Pi, S_e)$;
5. $\mathcal{C} := \mathsf{MaxClique}(G)$;
6. $\mathcal{A} := \mathsf{GetSug}(S_e, \mathcal{C}, \Omega(S_e), \mathsf{V}_\mathcal{B})$;
7. **return** $(\mathcal{A}, \mathsf{V}(\mathcal{A}))$;

Fig. 10. Algorithm Suggest

graph of Fig. 9. Observe the following. (a) The values of job, AC and zip depend on the value of status by rules n_1, n_2 and n_3, respectively. (b) The value of AC in turn decides city by n_5. (c) From city and zip one can derive county by n_4. Hence, the set of attributes that can be derived from clique $\mathcal{C}_1$ is $\mathcal{A}' = \{\mathsf{job}, \mathsf{AC}, \mathsf{zip}, \mathsf{city}, \mathsf{county}\}$. This yields a suggestion $(\mathcal{A}, \mathsf{V}(\mathsf{status}))$, where $\mathcal{A} = R \setminus (\mathcal{A}' \cup \mathcal{B}) = \{\mathsf{status}\}$, and $\mathsf{V}(\mathsf{status}) = \{$*retired*, *unemployed*$\}$. As long as users identify the true value of status, the true value of George exists, and can be automatically deduced as described in Example 9. □

However, a clique $\mathcal{C}$ and S_e may still have conflicts and as a result, $\mathcal{C}$ may not yield a valid completion of S_e, as illustrated by the example below.

Example 14: Consider the clique $\mathcal{C}_2$ shown in Fig. 9 with three nodes n_5, n_6 and n_8. Observe the following: (a) n_5 indicates that *312* $\prec^v_{\mathsf{AC}}$ *212*, since *212* is *assumed* the latest AC value; whereas (b) n_6, n_8 and constraint φ_6 in Fig. 3 state that *312* is the latest AC value, *i.e., 212* $\prec^v_{\mathsf{AC}}$ *312*. Therefore, the values embedded in clique $\mathcal{C}_2$ may not lead to a valid completion for entity instance E_2, *i.e.,* $\mathcal{C}_2$ and S_e have conflicts. □

To handle conflicts between $\mathcal{C}$ and S_e, we use MaxSat to find a maximum subgraph $\mathcal{C}'$ of $\mathcal{C}$ that has no conflicts with S_e (MaxSat is to find a maximum set of satisfiable clauses in a Boolean formula; see *e.g.,* [Selman and Kautz 2004]). For instance, for clique $\mathcal{C}_2$ of Example 14, we use a MaxSat-solver [Selman and Kautz 2004] to identify clique $\mathcal{C}'_2$ with nodes n_6 and n_8, which has no conflicts with the specification for George. We then derive $\mathcal{A}' = \{\mathsf{job}, \mathsf{zip}\}$ from $\mathcal{C}'_2$. Since $\mathcal{B}$ is $\{\mathsf{name}, \mathsf{kids}\}$ (Example 13), we find $\mathcal{A} = R \setminus (\mathcal{A}' \cup \mathcal{B}) = \{\mathsf{status}, \mathsf{city}, \mathsf{AC}, \mathsf{county}\}$ for a suggestion.

5.3.2. Computing Suggestions. We are now ready to present the algorithm for computing suggestions, referred to as Suggest, which is shown in Fig. 10. It takes as input a specification S_e of e, partial orders O_d deduced from S_e ($S_e \models O_d$, by Algorithm DeduceOrder), and the set $\mathsf{V}_\mathcal{B}$ of validated true values. It finds and returns a suggestion $(\mathcal{A}, \mathsf{V}(\mathcal{A}))$.

Algorithm Suggest first computes candidate true values for all attributes whose true values are yet unknown (line 1). It then deduces a set of derivation rules from the instance constraints $\Omega(S_e)$ (line 2) of S_e (line 3; as illustrated in Example 11). Based on these derivation rules, it builds a compatibility graph (line 4; see Example 12) and identifies a maximum clique $\mathcal{C}$ in the graph (line 5). Finally, it generates a suggestion using the clique (line 6; see Examples 13 and 14).

We next present the details of the procedures used in algorithm Suggest one by one.

DeriveVR. For each attribute A whose true value is yet unknown, it computes a set $\mathsf{V}(A)$ of candidate true values for A. Observe that given an attribute A, for any value a_1 in its active domain $\mathsf{adom}^v(A)$, if there exists another value a_2 also from $\mathsf{adom}^v(A)$,

Procedure TrueDer

Input: A specification S_e of an entity,
a set of instance constraints $\Omega(S_e)$, and
the candidate true values $\mathsf{V}(R)$.
Output: A set Π of derivation rules.

1. $\Pi := \emptyset$;
2. **for each** $\forall t\ (\nu \rightarrow t[A_r] = c_r) \in \Gamma$, $\nu = \bigwedge_{i=1}^{k} t[A_i] = c_i$ **do**
3. **if** for all attribute $A_i \in \mathcal{B}$, $c_i \in V(A)$ **then**
4. $\Pi := \Pi \cup \{(\{A_1, \ldots, A_k\}, \{c_1, \ldots, c_k\}) \rightarrow (A_r, c_r)\}$;
5. **for each** $B \in R$ **do**
6. **for each** $b \in V(B)$ **do**
7. $U_{(B,b)} := \{b_i \prec^v_B b \mid b_i \in \mathsf{V}(B) \setminus \{b\}\}$;
8. $\Omega_{(B,b)} := \{\phi \mid \phi \in \Omega(S_e) \wedge \phi = \omega \rightarrow b_i \prec^v_B b\}$;
9. $X := \emptyset$; $P(X) := nil$; success := *true*;
10. **for each** $b_i \in U_{(B,b)}$ **do**
11. pick a rule ϕ from $\Omega_{(B,b)}$ for $b_i \prec_B b$;
12. **if** $P(X)$ satisfies ϕ **then**
13. populate $X, P(X)$;
14. **else** success :=*false*; **break** ;
15. **if** success **then** $\Pi := \Pi \cup \{(X, P(X)) \rightarrow (B, b)\}$;
16. **return** Π;

Fig. 11. Procedure TrueDer

such that $a_1 \prec^v_A a_2$, then a_1 is not the true value for A. In other words, a_1 is known not to be the most current value.

Based on this observation, DeriveVR works as follows (not shown). Initially, $\mathsf{V}(A)$ takes the active domain $\mathsf{adom}^v(I_e.A)$. It then removes all values a_1 in $\mathsf{adom}^v(I_e.A)$ from $\mathsf{V}(A)$ for which there exists a value a_2 in $\mathsf{adom}^v(I_e.A) \setminus \{a_1\}$ such that $a_1 \prec^v_A a_2$ is in the deduced partial order O_d, as a_2 is more current than a_1 in A. DeriveVR takes $O(|I_t|^2)$ time with an index, since it checks at most $|O_d|$ partial orders, and $|O_d| \leq |I_t|^2$.

TrueDer. Given a set $\Omega(S_e)$ of instance constraints, procedure TrueDer deduces a set Π of derivation rules following the same way as shown in Example 11.

(1) From constant CFDs, the derivation rules could be deduced directly as long as they do not have conflicts with the candidate true values derived by DeriveVR.

(2) From those instance constraints that represent currency constraints and currency orders, it deduces derivation rules of the form $(X, P(X)) \rightarrow (B, b)$, for each attribute B whose true value is unknown and for each $b \in \mathsf{V}(B)$, if such a rule exists. While it is prohibitively expensive to enumerate all these rules, we use a heuristics to find a set of derivation rules. For each candidate true value b, it first identifies instance constraints that could complement the missing partial orders when assuming b as true value. Then it maintains and populates pattern $(X, P(X))$ from unknown attributes and candidate true values that could satisfy the premise of each of those instance constraints.

For example, the rule n_1 in Example 11 could be deduced as follows. Observe that with φ_5 (Fig. 3) and r_5, r_6 (Fig. 2), the instance constraint

$$(\text{unemployed} \prec^v_{\mathsf{status}} \text{retired} \rightarrow \text{n/a} \prec^v_{\mathsf{job}} \text{veteran})$$

could be derived. Here $\mathsf{V}(\mathsf{job})$ consists of 2 values "n/a" and "veteran". The users may inspect the two values and choose one from the two as the true value of $\mathsf{V}(\mathsf{job})$. If one wants to assume "veteran" as the true value, "n/a $\prec^v_{\mathsf{job}}$ veteran" is missing from the partial order. Nonetheless, this can be complemented with the instance constraint given above. In light of this, we populate X as $\{\mathsf{status}\}$, and $P(X)$ as "retired" ($\in \mathsf{V}(\mathsf{status})$) to

Procedure CompGraph

Input: A set Π of derivation rules, and a specification S_e.
Output: A compatibility graph G of Π.

1. initialize G to be an empty graph;
2. **for each** derivation rule $n \in \Pi$ **do**
3. add a node in G for n;
4. **for each** node $n_i \in G$ **do**
5. **for each** node $n_j \in G$ where $n_i \neq n_j$ **do**
6. **if** n_i and n_j are compatible **then**
7. add an edge (n_i, n_j) to G;
8. **return** G;

Fig. 12. Procedure CompGraph

satisfy the premise of the constraint. When $(X, P(X))$ is in place, the derivation rule n_1 : ({status}, {*retired*}) $\rightarrow$ (job, *veteran*) can be deduced.

More specifically, procedure TrueDer is given in Fig. 11. It works as follows, starting with an empty set Π of derivation rules (line 1).

(1) Deduce rules from CFDs: for each constant CFD $\forall t\ (\nu \rightarrow t[A_r] = c_r) \in \Gamma$ with $\nu = \bigwedge_{i=1}^{k} t[A_i] = c_i$, we check the following (line 2). If for each conjunct $t[A_i] = c_i$ in ν, with $A_i \in \mathcal{B}$, we have that $c_i \in \mathsf{V}(A)$, *i.e.,* when the values of the CFD have no conflict with those candidate true values (line 3), then we add $(\{A_1, \ldots, A_k\}, \{c_1, \ldots, c_k\}) \rightarrow (A_r, c_r)$ as a new derivation rule (line 4).

(2) Deduce rules from those instance constraints in $\Omega(S_e)$ that represent currency constraints and partial currency orders in S_e, as follows:
   (i) for each attribute B whose true value is unknown (line 5) and each value b in $\mathsf{V}(B)$ that can possibly be its true value (line 6), let $U_{(B,b)} = \{b_i \prec_B^v b \mid b_i \in \mathsf{V}(B) \setminus \{b\}\}$, which is the set consisting of all the missing partial orders when b is assumed to be the true value of B (line 7);
   (ii) partition instance constraints based on $U_{(B,b)}$: for each value b in $U_{(B,b)}$, let $\Omega_{(B,b)}$ consist of all instance constraints $\phi \in \Omega(S_e)$ such that ϕ is of the form $\omega \rightarrow b_i \prec_B^v b$ (line 8); note that each ϕ appears in at most one of the partitions;
   (iii) for each $b_i \in U_{(B,b)}$ (line 10), we pick a rule $\phi = \omega \rightarrow b_i \prec_B^v b$ from $\Omega_{(B,b)}$ (line 11); we then expand X and the pattern $P(X)$ so that the premise ω can be satisfied (line 13), until either $(X, P(X))$ can no longer satisfy ω (success = *false*) (line 14), or each $b_i \prec_B^v b$ in $U_{(B,b)}$ is covered by such a rule ϕ (success = *true*); in the latter case, we add the rule $(X, P(X)) \rightarrow (B, b)$ to Π (line 15). Note that $|X| \leq |R|$.

The procedure is in $O((|\Sigma|+|\Gamma|)|I_t|^2+|I_t|^3)$ time. Indeed, the cost of step (1) is bounded by $O(|\Gamma|)$; and for step (2), since $U_{(B,b)}$'s are disjoint, $\Omega_{(B,b)}$'s partition $\Omega(S_e)$, and moreover, each ϕ in $\Omega(S_e)$ is used at most once, it takes at most $O((|\Sigma| + |\Gamma|)|I_t|^2 + |I_t|^3)$ time.

CompGraph. Given a set of derivation rules, procedure CompGraph generates their compatibility graph $G(N, E)$ (see Example 12 for a running example).

More specifically, CompGraph is presented in Fig. 12. It takes a set Π of derivation rules as input. It constructs and returns a compatibility graph for Π. The procedure works as follows. It first initializes a compatibility graph (line 1). It then generates a node for each derivation rule (line 2). The edges are then added (lines 3-6). For any two distinct nodes, if their associated rules are compatible (line 5, see the definition of compatibility graphs given earlier), an edge is added to connect these two nodes (line 6). It terminates and returns a compatibility graph G (line 7).

Procedure GetSug

Input: A specification S_e, instance constraints $\Omega(S_e)$, a clique $\mathcal{C}$, and $\mathsf{V}_{\mathcal{B}}$.
Output: A set $\mathcal{A}$ of attributes as suggestion.

1. $\mathcal{A}' := R; P'[\mathcal{A}'] := nil$;
2. **for each** $(X, P(X)) \rightarrow (B, b) \in \mathcal{C}$ **do**
3. $\quad \mathcal{A}' := \mathcal{A}' \setminus \{B\}$
4. $\quad P'[X] := P(X)$;
5. instantiate P' and convert P' to CNF Φ';
6. $\Phi_u := \Phi' \cup \Phi(S_e)$;
   /* each clause in Φ' is assigned with a weight of 1.0, and each clause in $\Phi(S_e)$ is assigned with a weight of $+\infty$.*/
7. $\Phi_s :=$ Weighted-MAXSAT (Φ_u);
8. convert Φ_s to P'';
9. $\mathcal{A} := R \setminus \mathsf{V}_{\mathcal{B}}$;
10. **for each** $(X, P(X)) \rightarrow (B, b) \in \mathcal{C}$ **do**
11. $\quad$ **if** $P(X) \neq P''[X]$ **then** remove $(X, P(X)) \rightarrow (B, b)$ from $\mathcal{C}$;
12. $\quad$ **else** $\mathcal{A} := \mathcal{A} \setminus \{B\}$;
13. **return** $\mathcal{A}$;

Fig. 13. Procedure GetSug

It is readily to verify that the procedure takes at most $O(|\Pi|^2)$ time, where $|\Pi|$ is no larger than $|R||I_t|$.

MaxClique. Given a compatibility graph $G(N, E)$, this procedure computes a maximum clique $\mathcal{C}$ of $G(N, E)$. While it is intractable to find a maximum clique, several tools have been developed for computing maximum cliques, with a good approximation bound (*e.g.,* [Feige 2005]). We employ one of these tools as MaxClique.

GetSug. Given a specification S_e of an entity, a set $\Omega(S_e)$ of instance constraints, a clique $\mathcal{C}$, and $\mathsf{V}_{\mathcal{B}}$, the procedure computes a suggestion as output. As shown in Examples 13 and 14, the clique returned by MaxClique represents a suggestion, but the suggestion may contain conflicts. This procedure is to convert the clique to a suggestion, and revise it in the presence of conflicts by invoking a weighted MaxSat-solver.

More specifically, GetSug is given in Fig. 13. It works as follows. It first identifies the required attributes $\mathcal{A}'$ and pattern $P'[\mathcal{A}]$ by applying derivation rules in $\mathcal{C}$ (lines 1-4). It then converts P' to a CNF Φ', along the same line as procedures Instantiation and ConvertToCNF given earlier (line 5). Since Φ' may have conflicts with the $\Phi(S_e)$, it invokes a weighted MAXSAT-solver to minimally revise Φ' such that $\Phi' \cup \Phi(S_e)$ is satisfiable (lines 6-7). It then finds the subset of $\mathcal{C}$ corresponding to the revised Φ', which has no conflicts with S_e (lines 8-12). It also derives a set $\mathcal{A}$ of attributes from the subset of $\mathcal{C}$ (line 12; see Example 13). Finally, it returns $\mathcal{A}$ (line 13). Recall that $\mathsf{V}(A)$ is computed by procedure DeriveVR given earlier. Note that the input to the MaxSat-solver is no larger than $|R|^2|I_t|^2$. Moreover, there are efficient MaxSat-solvers available, with a reasonable approximation bound [Selman and Kautz 2004].

Correctness. Algorithm Suggest guarantees to generate a suggestion $(\mathcal{A}, \mathsf{V}(\mathcal{A}))$. Indeed, (1) the clique $\mathcal{C}'$ revised by MaxSat has no conflicts with S_e, and thus $\mathcal{C}'$ and S_e warrant to have a valid completion I_t^c. Let $t_c = \mathsf{LST}(I_t^c)$. If $\mathsf{V}(\mathcal{A})$ are validated for $\mathcal{A}$, then t_c must be the *true value* $\mathsf{T}(S_e)$ of S_e, since $t_c[\mathcal{B}] = \mathsf{V}_{\mathcal{B}}$ remains unchanged for all valid completions of S_e, and $t_c[\mathcal{A}']$ is uniquely determined by $t_c[\mathcal{A}]$ and $\mathsf{V}_{\mathcal{B}}$ by the construction. (2) All possible true values for $\mathcal{A}$ from their active domains are already included in $\mathsf{V}(\mathcal{A})$.

6. EXPERIMENTAL STUDY

We conducted experiments using both real-life and synthetic data. We evaluated the accuracy and scalability of (1) IsValid for validating a specification, (2) DeduceOrder for deducing true values, (3) Suggest for computing suggestions, and (4) the overall performance of conflict resolution supporting (1-3) above. Note that IsValid and DeduceOrder are useful in their own right, since users may want to check their specifications and infer true values outside the interaction framework.

Experimental data. We used two real-life datasets (NBA and CAREER) and synthetic data (Person). Constraints were discovered semi-automatically with the help of profiling algorithms [Chiang and Miller 2008; Calders et al. 2006] (see details below). Timestamps for the datasets were either missing (for CAREER and Person) or incomplete (NBA). We assumed *empty currency orders* in all the experiments even when partial timestamps were given. The true values of these entities were first selected using (incomplete) timestamps, and then extracted and examined manually. Also, the available (incomplete) timestamps were used for designing currency constraints.

NBA *player statistics*. This dataset was retrieved from the following sites: (1) *http://databasebasketball.com/*, (2) *http://www.infochimps.com/marketplace*, and (3) *http://en.wikipedia.org/wiki/List_of_National_Basketball_Association_arenas*. It consists of three tables: (a) Player (from sources 1 and 3) contains information about players, identified by player id (pid). (b) Stat (from 1) includes the statistics of these players from the 2005/2006 season to the 2010/2011 season. (c) Arenas (from 3) records the historical team names and arenas of each team.

We created a table, referred to as NBA, by first joining Player and Stat via *equi-join* on the pid attribute, and then joining Arenas via *equi-join* on the team attribute. The NBA table consists of 19573 tuples for 760 entities (*i.e.,* players). Its schema is (pid, name, true name, team, league, tname, points, poss, allpoints, min, arena, opened, capacity, city). When producing the NBA table we took care of the attributes containing multiple values for a player, *e.g.,* multiple teams for the same player, and multiple teams for one arena. We ensure that only one attribute value (*e.g.,* team) appears in any tuple. Only data from (1) and (3) carries (partial) timestamps. Therefore, the true values of entities in the NBA table *cannot* be directly derived when putting (1), (2) and (3) together.

The number of tuples pertaining to an entity ranges from 2 to 136, about 27 in average. We consider *entity instances*, *i.e.,* tuples referring to the same entity, which are *much smaller* than a database.

We found 54 currency constraints: 15 for team names (tname) as shown by φ_1 below; 32 for arena, similar to φ_2; and 4 (resp. 3) for attribute allpoints that were scored since 2005 (resp. arena), similar to φ_3 (resp. φ_4), where B ranges over points, poss, min and tname (resp. opened, capacity and years). We deduced 58 constant CFDs, *e.g.,* ψ_1 below.

φ_1: $\forall t_1, t_2$ (t_1[tname] = "New Orleans Jazz" $\wedge$ t_2[tname] = "Utah Jazz" $\rightarrow$ $t_1 \prec_{\text{tname}} t_2$);
φ_2: $\forall t_1, t_2$ (t_1[arena] = "Long Beach Arena" $\wedge$ t_2[arena] = "Staples Center" $\rightarrow$ $t_1 \prec_{\text{arena}} t_2$);
φ_3: $\forall t_1, t_2$ (t_1[allpoints] $< t_2$[allpoints] $\wedge$ $t_1[B] \neq t_2[B] \rightarrow t_1 \prec_{\text{B}} t_2$)
φ_4: $\forall t_1, t_2$ ($t_1 \prec_{\text{arena}} t_2 \wedge t_1[B] \neq t_2[B] \rightarrow t_1 \prec_{\text{B}} t_2$)
ψ_1: $\forall t$ (t[arena] = "United Center" $\rightarrow$ t[city] = "Chicago, Illinois")

(2) CAREER. This data set was retrieved *as is* from the following source: *http://www.cs.purdue.edu/commugrate/data/citeseer*. Its schema is (first name, last name, affiliation, city, country). We chose 65 persons from the dataset, and for each one, we collected all of his/her publications, one tuple for each. *No reliable timestamps* were available for this dataset. The number of tuples pertaining to an entity ranges from 2 to 175, about 32 in average.

We derived 503 currency constraints: if two papers A and B are by the same person and A cites B, then the affiliation and address (city and country) used in paper A are more current than those used in paper B. We also found a single CFD of the form: $\forall t\ (t[\mathsf{affiliation}] = c_1 \rightarrow t[\mathsf{city}] = c_2 \wedge t[\mathsf{country}] = c_3)$, with 347 patterns with different constants (c_1, c_2, c_3).

The constraints for each dataset (NBA and CAREER) have essentially *the same form*, and *only differ* in their constants. Indeed, we find that the number of constraints with different forms is rather *small* in practice.

(3) Person *data*. The synthetic data adheres to the schema given in Table 2. We found 983 currency constraints (of *the same form* but with distinct constant values for status, job and kid) and a single CFD $\forall t\ (t[\mathsf{AC}] = c_1 \rightarrow t[\mathsf{city}] = c_2)$ with 1000 patterns (c_1, c_2) (counted as distinct constant CFDs), similar to those in Table 3. The data generator used two parameters: n denotes the number of entities, and s is the size of *entity instances* (the number of tuples pertaining to an entity). For each entity, it first generated a true value t_c, and then produced a set E of tuples that have conflicts but do not violate the currency constraints; we treated $E \setminus \{t_c\}$ as the entity instance. We generated n = 10k entities, with s from 1 to 10k. We used *empty* currency orders.

Discovery of currency constraints. For each dataset, the currency constraints were discovered in the following 3 steps. We first sampled a small number of tuples from each dataset, and manually tagged the currency orders of values in the sampled data. Second, along the same line of profiling algorithms [Chiang and Miller 2008; Calders et al. 2006], candidate constraints were deduced from the sampled data. Finally, we manually examined these candidates to find common patterns that may hold on the rest of dataset. Following the common patterns, more constraints could be discovered by propagating these patterns with values from the rest of the dataset.

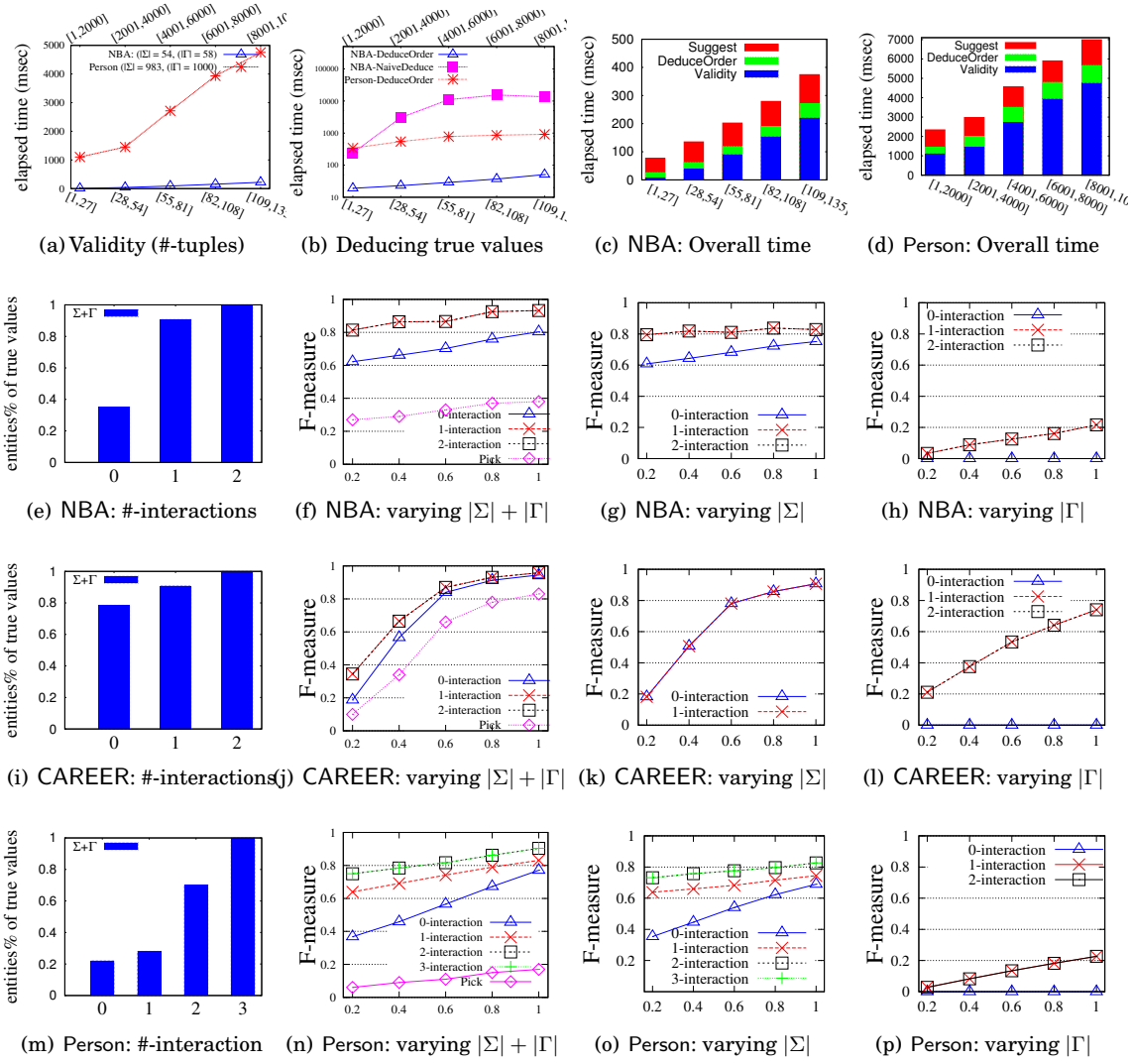
Algorithms. We implemented the following algorithms in Python: (a) IsValid (Section 5.1): it calls MiniSat [Giunchiglia and Tacchella 2004] as the SAT-solver; (b) DeduceOrder and NaiveDeduce, where NaiveDeduce repeatedly invokes MiniSat [Giunchiglia and Tacchella 2004], as described in Section 5.2; and (c) Suggest: it uses MaxClique [Feige 2005] to find a maximal clique, and MaxSat-solver [Selman and Kautz 2004] to derive a suggestion (Section 5.3). We simulated user interactions by providing true values for suggested attributes, some with new values, *i.e.,* values not in the active domain. We also implemented (d) Pick, a traditional method that randomly takes a value [Bleiholder and Naumann 2008] (*a.k.a.* "roll the dice" strategy). Pick works as follows. Instead of picking a value from all possible values, it first generates a set of candidate current values as given by the currency constraints, *i.e.,* the values that are not less current than any other value. It then randomly picks a value from the candidate current values, as the true value.

Accuracy. To measure the quality of suggestions, we used the notion of F-measure (*http://en.wikipedia.org/wiki/F-measure*):

$$\mathsf{F\text{-}measure} = 2 \cdot (\mathsf{recall} \cdot \mathsf{precision})/(\mathsf{recall} + \mathsf{precision}).$$

Here precision is the ratio of the number of values correctly deduced to the total number of values deduced; and recall is the ratio of the number of values correctly deduced to the total number of attributes with conflicts or stale values.

Implementation. Our algorithms were implemented in Python. All experiments were conducted on a Linux machine with a 3.0GHz Intel CPU and 4GB of Memory. Each experiment was repeated 5 times, and the average is reported here.



(a) Validity (#-tuples) (b) Deducing true values (c) NBA: Overall time (d) Person: Overall time

(e) NBA: #-interactions (f) NBA: varying $|\Sigma| + |\Gamma|$ (g) NBA: varying $|\Sigma|$ (h) NBA: varying $|\Gamma|$

(i) CAREER: #-interactions (j) CAREER: varying $|\Sigma| + |\Gamma|$ (k) CAREER: varying $|\Sigma|$ (l) CAREER: varying $|\Gamma|$

(m) Person: #-interaction (n) Person: varying $|\Sigma| + |\Gamma|$ (o) Person: varying $|\Sigma|$ (p) Person: varying $|\Gamma|$

Fig. 14. Experimental results

Experimental results. We next present our findings. Due to the small size of the CAREER data for each entity, experiments conducted on it took typically less than 10 milliseconds (ms). Hence we do not report its result in the efficiency study.

Exp-1: Validity checking. We first evaluated the scalability of IsValid. The average time taken by entity instances of various sizes is reported in Fig. 14(a), where the lower x-axis shows the sizes of NBA, and the upper x-axis is for Person data. The results show that IsValid suffices to validate specifications of a reasonably large size. For example, it took 220 ms for NBA entity instances of 109-135 tuples and 112 constraints, with 14 attributes in each tuple. For Person, it took an average of 4.7 seconds on entities of 8k-10k tuples and 1983 constraints. We also find IsValid accurate: specifications reported (in)valid are indeed (in)valid.

Exp-2: Deducing true values. We next evaluated the performance of algorithms DeduceOrder and NaiveDeduce. The results on both NBA and Person data are reported in Fig. 14(b), which tell us the following: (a) DeduceOrder scales well with the size of entity instances, and (b) DeduceOrder substantially outperforms NaiveDeduce on both datasets, for reasons given in Section 5.2. Indeed, DeduceOrder took 51 ms on NBA entity instances with 109-135 tuples, and 914 ms on Person entities of 8k-10k tuples; in contrast, NaiveDeduce spent 13585 ms and over 20 minutes (hence not shown in Fig. 14(b)) on the same datasets, respectively.

We also find that DeduceOrder derived as many true values as NaiveDeduce on both datasets (not shown). This tells us that DeduceOrder can efficiently deduce true values on large entity instances without compromising the accuracy of the true values found.

Furthermore, we also learned from our experiments that true values were often scattered across multiple tuples. For example, for the NBA data, only 31% of true values were deduced from a single tuple.

Exp-3: Suggestions for user interactions. We evaluated the accuracy of suggestions generated from currency constraints Σ and CFDs Γ put together. The results on NBA, CAREER and Person are given in Figures 14(e), 14(i) and 14(m), respectively, where the x-axis indicates the rounds of interactions, and the y-axis is the percentage of true attribute values deduced. When choosing a fraction of constrains in this experiment, we used random selection. As in all the experiments of this paper, we report the average result over 5 runs of the experiment.

The results tell us the following. (a) Few rounds of interactions are needed to find all the true attribute values for an entity: at most 2, 2 and 3 rounds for NBA, CAREER and Person data, respectively. (b) A large part of true values can be *automatically deduced* by means of currency and consistency inferences: 35%, 78% and 22% of true values are identified from $\Sigma + \Gamma$ *without user interaction*, as indicated by the 0-interaction in Figures 14(e), 14(i) and 14(m), respectively.

Impact of $|\Sigma|$ *and* $|\Gamma|$. To be more precise when evaluating the accuracy, we use F-measure, which combines precision and recall, and take the cases of using $|\Gamma|$ only or $|\Sigma|$ only into consideration. Figures 14(f)–14(h), 14(j)–14(l) and 14(n)–14(p) show the results for NBA, CAREER and Person, respectively, when varying both $|\Sigma|$ and $|\Gamma|$, $|\Sigma|$ only, and varying $|\Gamma|$ alone, respectively. The x-axis shows the percentage of Σ or Γ used, and the y-axis shows the corresponding F-measure values.

These results tell us the following. (a) As shown in Figures 14(f), 14(j) and 14(n), our method substantially outperforms the traditional method Pick, by 201% in average on all datasets, even when we favor Pick by allowing it to capitalize on currency orders. This verifies that data currency and consistency can significantly improve the accuracy of conflict resolution. (b) When Σ and Γ are taken together, the F-measure value is up to 0.930 for NBA (Fig. 14(f), the top right point), 0.958 for CAREER (Fig. 14(j)), and 0.903 for Person (Fig. 14(n)), in contrast to 0.830 in Fig. 14(g), 0.907 in Fig. 14(k), and 0.826 in Fig. 14(o), respectively, when Σ is used alone, and as opposed to 0.210 in Fig. 14(h), 0.741 in Fig. 14(l), and 0.234 in Fig. 14(p), respectively, with Γ only. These further verify that the inferences of data currency and consistency should be *unified* instead of taken separately. (c) The more currency constraints and/or CFDs are available, the higher the F-measure is, as expected. (d) The two curves for the 2- and 1-interaction overlap in Figures 14(f)–14(h) for NBA, 2- and 1-interaction in Figures 14(j)–14(l) for CAREER, and 3- and 2-interaction in Figures 14(n)–14(p) for Person. These indicate that the user interactions are needed to provide true values for those attributes that we do not have enough information to deduce their true values.

Exp-4: Efficiency. The overall performance for resolving conflicts in the NBA (resp. Person) data is reported in Fig. 14(c) (resp. Fig. 14(d)). Each bar is divided into the elapsed time taken by (a) validity checking, (b) true value deducing, and (c) suggestion generating, including computing maximal cliques and running MaxSat. The result shows that conflict resolution can be conducted efficiently in practice, *e.g.,* each round of interactions for NBA took 380 ms. Here validating specifications takes most time, dominated by the cost of SAT-solver, while deducing true values takes the least time.

Summary. From the experimental results we find the following. (a) Conflict resolution by reasoning about data currency and consistency substantially outperforms the traditional method Pick, by 201%. (b) It is more effective to unify the inferences of data currency and consistency than treating them independently. Indeed, when Σ and Γ are taken together, the F-measure improves over Σ only and Γ only by 11% and 236%, respectively. (c) Our conflict resolution method is efficient: it takes less than 0.5 second on the real-life datasets even with interactions. (d) Our method scales well with the size of entities and the number of constraints. Indeed, it takes an average of 7 seconds to resolve conflicts in Person entity instances of 8k-10k tuples, with 1983 constraints. (e) At most 2-3 rounds of interactions are needed for all datasets.

7. CONCLUSION

We have proposed a model for resolving conflicts in entity instances, based on both data currency and data consistency. We have also identified several problems fundamental to conflict resolution, and established their complexity. Despite the inherent complexity of these problems, we have introduced a framework for conflict resolution, along with practical algorithms supporting the framework. Our experimental study has verified that our methods are effective and efficient, using real-life and synthetic data. We contend that these yield a promising approach to resolving conflicts in practice.

Several topics are targeted for future work. We are now exploring efficient algorithms with better performance guarantees for generating suggestions, and testing them with data in various domains. Another topic concerns the discovery of data quality rules. Prior work on discovery of such rules [Chiang and Miller 2008] shows that a large number of high-quality rules can be identified from possibly dirty data. A third topic is to repair data by using currency constraints and partial temporal orders. This is more challenging than conflict resolution, since a database to be repaired is typically *much larger* than entity instances. Finally, a challenging topic is to extend our framework by allowing users to edit constraints, and by soliciting other information (such as semantic dependencies specifying how attributes are correlated) as users' feedback when the users do not have sufficient currency knowledge about their data.

Acknowledgment. Fan and Yu are supported in part by 973 Programs 2012CB316200 and 2014CB340302, NSFC 61133002, Guangdong Innovative Research Team Program 2011D005, Shenzhen Peacock Program 1105100030834361, and EPSRC EP/J015377/1.

REFERENCES

Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*.

Leopoldo Bertossi. 2011. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers.

Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (Eds.). 2009. *Handbook of Satisfiability*. Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press.

Jens Bleiholder and Felix Naumann. 2008. Data fusion. *ACM Comput. Surv.* 41, 1 (2008).

Toon Calders, Bart Goethals, and Szymon Jaroszewicz. 2006. Mining rank-correlated sets of numerical attributes. In *KDD*.

Fei Chiang and Renee Miller. 2008. Discovering Data Quality Rules. In *VLDB*.

Jan Chomicki and David Toman. 2005. Time in Database Systems. In *Handbook of Temporal Reasoning in Artificial Intelligence*. Elsevier.

Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving Data Quality: Consistency and Accuracy. In *VLDB*.

Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *SIGMOD*.

Umeshwar Dayal. 1983. Processing Queries Over Generalization Hierarchies in a Multidatabase System. In *VLDB*.

Xin Dong, Laure Berti-Equille, and Divesh Srivastava. 2009. Truth Discovery and Copying Detection in a Dynamic World. In *VLDB*.

Xin Dong and Felix Naumann. 2009. Data fusion - Resolving Data Conflicts for Integration. In *VLDB*.

W. W. Eckerson. 2002. Data quality and the bottom line: Achieving business success through a commitment to high quality data. *The Data Warehousing Institute* (2002).

Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate Record Detection: A Survey. *TKDE* 19, 1 (2007).

Wenfei Fan and Floris Geerts. 2012. *Foundations of Data Quality Management*. Morgan & Claypool Publishers.

Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional Functional Dependencies for Capturing Data Inconsistencies. *TODS* 33, 1 (2008).

W. Fan, F. Geerts, J. Li, and M. Xiong. 2011. Discovering Conditional Functional Dependencies. *TKDE* 23, 5 (2011), 683–698.

Wenfei Fan, Floris Geerts, Ma Shuai, Nan Tang, and Wenyuan Yu. 2013. Inferring Data Currency and Consistency for Conflict Resolution. In *ICDE*.

Wenfei Fan, Floris Geerts, and Jef Wijsen. 2012. Determining the currency of data. *TODS* 37, 4 (2012).

Uriel Feige. 2005. Approximating Maximum Clique by Removing Subgraphs. *SIAM J. Discret. Math.* 18 (February 2005). Issue 2.

Alban Galland, Serge Abiteboul, Amélie Marian, and Pierre Senellart. 2010. Corroborating information from disagreeing views. In *WSDM*.

Enrico Giunchiglia and Armando Tacchella (Eds.). 2004. *Theory and Applications of Satisfiability Testing*.

Bart Goethals. 2003. *Survey on Frequent Pattern Mining*. Technical Report. Helsinki Institute for Information Technology.

Rob Goldring. 1995. Update Replication: What Every Designer Should Know. In *InfoDB, Vol.9, No.2*. 17–24.

Sergio Greco, Cristina Sirangelo, Irina Trubitsyna, and Ester Zumpano. 2003. Preferred Repairs for Inconsistent Databases. In *IDEAS*.

Pei Li, Xin Dong, Andrea Maurino, and Divesh Srivastava. 2011. Linking Temporal Records. *PVLDB* (2011).

Amihai Motro and Philipp Anokhin. 2006. Fusionplex: resolution of data inconsistencies in the integration of heterogeneous information sources. *Information Fusion* 7, 2 (2006).

Christos H Papadimitriou. 1994. *Computational Complexity*. Addison Wesley.

Bart Selman and Henry Kautz. 2004. Walksat Home Page. (2004). *http://www.cs.washington.edu/homes/kautz/walksat/*.

Larry J. Stockmeyer. 1976. The Polynomial-Time Hierarchy. *Theore. Comput. Sci* 3, 1 (1976).

Jennifer Widom. 2005. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In *CIDR*.

Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, and Mourad Ouzzani. 2010. GDR: a system for guided data repair. In *SIGMOD*.

Xiaoxin Yin, Jiawei Han, and Philip S. Yu. 2008. Truth Discovery with Multiple Conflicting Information Providers on the Web. *TKDE* 20, 6 (2008).

Haopeng Zhang, Yanlei Diao, and Neil Immerman. 2010. Recognizing Patterns in Streams with Imprecise Timestamps. In *VLDB*.